# A PC BASED LAN MONITOR

A Thesis Submitted
In Partial fulfilment of the Requirements
for the Degree of

**MASTER OF TECHNOLOGY**

*by*

K. SHANKAR

*to the*

**DEPARTMENT OF ELECTRICAL ENGINEERING**
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

FEBRUARY, 1989

6/2/89

## CERTIFICATE

Certified that the work entitled 'A PC BASED LAN MONITOR' has been carried out under my supervision and has not been submitted elsewhere for a degree.

K R Srivathsan

Dr. K. R. SRIVATHSAN
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology
KANPUR - 208 016

## ACKNOWLEDGEMENTS

I wish to express my deep gratitude and sincere thanks to Dr. K. R. SRIVATHSAN for suggesting the topic and for his invaluable guidance throughout the project, and to Dr. S. K. Bose for his guidance in the initial stages of the project.

I thank Mr. Sudhakar, Mr. George Joy, Mr.R. GopalKrishna and other Research Engineers for their their cooperation and timely help in the completion of this project.

Finally, I would like to thank my friends for their assistance throughout the project.

K . SHANKAR

## ABSTRACT

A Network Monitor is designed and implemented with a Personal Computer (PC) to monitor the traffic on a Token Ring network developed at IIT Kanpur. A layered approach is employed in the development of software which involves a Data Acquisition Layer that generates status messages for each packet circulated on the network and transfers them to the PC, a Data Processing Layer that processes the status messages and computes the various parameters, and a User Interface Layer that reports the network activity through network status displays.

# CONTENTS

# CHAPTER 1

# INTRODUCTION

A communication network when installed, should continue to function without intervention. However, no matter how reliable a component is ,it can still fail, resulting in communication failure or degradation of performance. Also, networks grow and change with the needs of users. Each network is originally designed to handle a certain level of throughput. All original assumptions are likely to change over time. This means that network design needs to be re-evaluated periodically. It is therefore important to manage the network and its growth so that the desired performance of the network is maintained despite failures and changing user needs. The quantity and geographical distribution of network devices makes the role of Network Management a difficult one. One of the important tools in managing networks is the Network Monitor.

A Network Monitor is a window to the network that provides continuous measurement of significant network performance indicators. The performance parameters obtained would help in fault diagnosis and also serve as inputs to prediction tools such as queuing models, statistical techniques, simulation and bench marks that would facilitate

efficient Network Management.

## 1.1 Types of Network Monitoring.

Depending on the method of measurement, Network Monitoring can be classified into two categories[KP 87]:

(1) Hardware Monitoring.

(2) Software Monitoring.

Hardware Monitoring is based on the assumption that the performance characteristics of computer systems can be measured by detecting signal voltage transitions in the circuitry of the hardware. Hardware monitoring could further be classified into two categories:

(a) Digital Monitoring - Involves signal monitoring at interface between data terminal equipment(DTE) and data communication equipment (DCE). Monitoring the interface signals between a modem and connected terminal or system is an example of Digital Monitoring.

(b) Analogue Monitoring - Involves direct measurement of analogue circuit conditions such as signal levels, phase jitter and distortion.

Software Monitors constitute a class of programs for measuring hardware , system and application software. They are resident in storage and can be activated by events or a timer. The data collected by sampling or eventing are reduced, processed and reported. State-of-the-art software

monitors maintain measurement data files or even data bases, which can be accessed at any time. Furthermore, threshold may be defined and thus exceptions, alarms and/or reports can be generated automatically. Software level monitoring techniques allow the measurement of the performance, error rates and also observe trends .It also has the advantage that additional hardware is not needed for the purpose of monitoring. In the description henceforth, Software Monitoring is implied, unless explicitly stated otherwise.

A Network Monitor could be functionally 'passive' or 'active'. When a Network Monitor is passive , it is not addressable by any other node on the network, and therefore, no other node sends information addressed to Network Monitor or vice versa. It functions as a listener transparent to the other network users and usually receives data by a T - connection on the communication channel. Active Network Monitoring is more advanced and sophisticated in that, different acquisition units could exist at various important subsystems in a network and report to the Network Monitor exceptions or performance information of the subsystem periodically. The Network Monitor may also send packets requesting stations to send specific information. In such a case the monitoring information transmitted from the data acquisition unit to the Network Monitor or vice versa would be an additional load on the Network. This would ,however,

provide a very useful tool in fault diagnosis and performance evaluation of different subsystems in large networks.

## 1.2 Fault Diagnosis with a Network Monitor.

Without the help of a Network Monitor, problem determination procedures are crude. If only one user complains, it is probably due to a problem with a station, or its software. If users of a specific server complain , it is most likely a server problem. If lots of users complain, however, the problem may be serious. A Network Monitor aids in the determination of the location of the fault to a great extent. The following are some of the common problems in networks, and the role of a Network Monitor in the corresponding fault diagnosis[JL 88].

(a) Problem: No network communications.

The Network Monitor would report throughput and packets per second as zero, showing no sign of traffic. This can be confirmed by attempting to transmit packets from a given node. If the problem is due to a short circuit, or open circuit, then ,it would not be successful. The best approach to track down the exact source would then be, to use a Time Domain Reflectometer(TDR).

(b)Problem: Abnormal Network Traffic.

This cannot be detected easily without a Network Monitor. The other possible symptoms are, large number of packets

with CRC error, and large amount of packets per second, particularly, of small packets that are smaller than the permissible length.The cause could be a malfunctioning CSMA or IBM Token Ring implementation.

(c) Problem: Station not communicating.

If a particular station, is unable to communicate with any other station ,then ,with the help of a Network Monitor it is possible to determine whether the problem is of the station alone or of a portion of the network.

(d) Problem: Station not able to communicate with some other stations.

The Network Monitor would be an invaluable tool in such cases. It would be possible to determine whether a malfunction occurred at the source (then, no packets would be reported to have been sent by the source node) , in the channel(error packets would be reported or alternatively no packets would be reported) or at the destination. The Network Monitor ,can, thus prove to be a useful tool in detecting failures.

## 1.3 Organisation of the Thesis.

In this project, a software monitor has been designed and implemented to monitor the performance of the PC - LAN developed at Indian Institute of Technology(IIT), Kanpur.

In the next chapter, a discussion of the design of Network Monitor is presented and a layered approach is suggested in the development of software. Chapters three,four and five discuss the implementation details of the layers of software. The concluding chapter discusses the performance of the Network Monitor and explores the enhancements and additional features that could be incorporated into the software.

# CHAPTER 2

# DESIGN OF NETWORK MONITOR

On-line analysis of data in communication networks involves acquisition of data, the subsequent processing of data, updating of data bases and reporting to the user the various parameters through user-friendly displays. In a network, data is generated in real time. On account of the time critical nature of execution of the monitor's tasks, it is imperative that the software is fast and efficient. In addition ,the software should preserve the flexibility to incorporate new features as and when necessary.

## 2.1 Design Criteria for a Network Monitor.

The general architecture of a Network Monitor is given in figure 2.1. The Network Monitor CPU controls all measurement and reporting functions. The addressable modules are responsible for gathering information on the network interface. The primary design and implementation criteria for the various modules of figure 2.1 may be broadly grouped as[KP 87]:

(1) Data acquisition.

(2) Centralised information availability.

(3) Data Base Management.

(4) Trend and performance analysis.

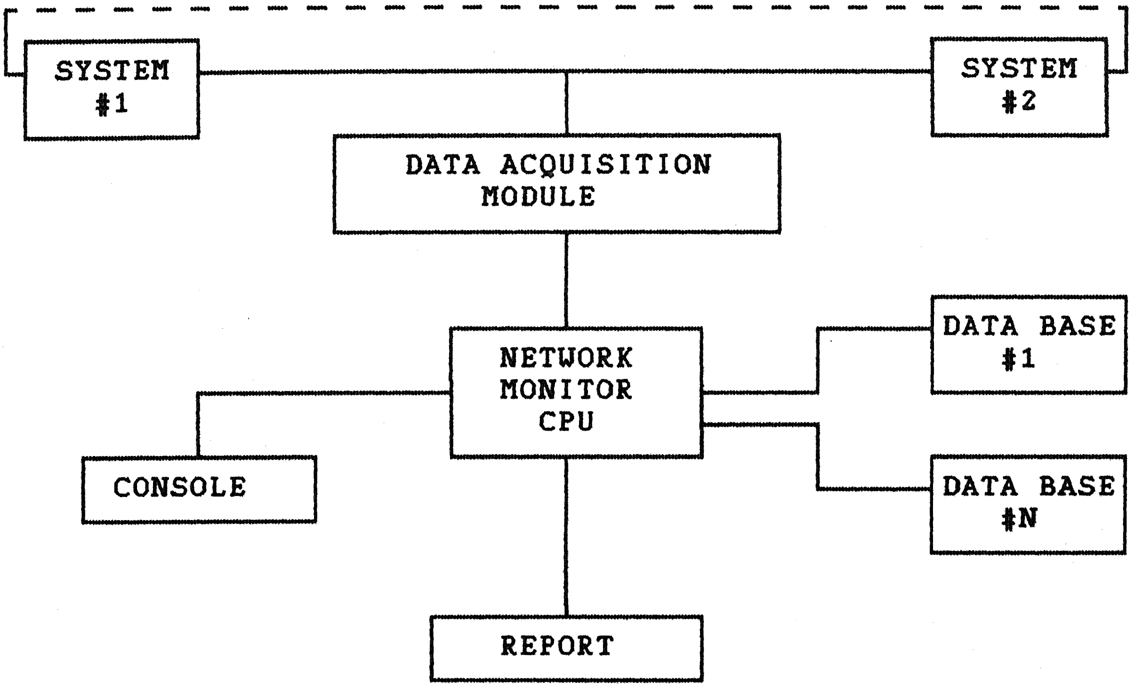

SYSTEM
#1
SYSTEM
#2
DATA ACQUISITION
MODULE
NETWORK
MONITOR
CPU
DATA BASE
#1
CONSOLE
DATA BASE
#N
REPORT

FIGURE 2.1

(5) Ease of use.

These criteria are briefly discussed in the the sub - sections that follow.

### 2.1.1 Data Acquisition.

The data acquisition modules communicate with the Network CPU through asynchronous communication ports. Once events of importance have occurred, the data acquisition module prepares status or alert messages and buffers them for transmission to the Network Monitor CPU. The Data Acquisition module should possess enough memory to buffer all the status messages when the load on the network is maximum. The transmission to the Network Monitor CPU should be such that there is no buffer overflow or overrun error in the status messages during this process. The modules for data acquisition are microprocessor based which include a processor, control logic and internal memory that contains programs to drive the processor and to buffer data. The programs are dependent on the protocol.

### 2.1.2 Centralised Information Availability.

It is important that information of the network be centrally available. The principal elements of the software should include the following:

a) Gathering of summary -type performance data from the data collection modules.

b) Maintaining summary tables on status and performance of every node in the network.

c) Updating graphic output to CRT displays and printers.

d) Processing commands of system users.

e) Recording performance statistics.

f) Providing access to the data bases for real time displays and printing reports.

### 2.1.3 Data Base Management.

In order to define meaningful performance optimisation, historical data should be available. The data stored in a performance data base can be interrogated and processed at will. Contents of the network performance data base may include service -oriented measures such as availability, response time, accuracy, and efficiency related measures such as Throughput and Utilisation. This would require the parameters to be collected, analysed and compressed at different levels of detail with date and time stamping. On account of the large amounts of data to be stored in the Data Base at regular intervals of time, it is vital that the update time is minimal, and therefore is an important criterion in the selection of the Data Base package for logging network data.

### 2.1.4 Trend and Performance Analysis.

Usually, data of the performance data base are further

processed, analysed, interpreted and reported using statistical and graphic features. Trend analysis can be used to track potential problems in the network configuration, suggesting the right places for reconfiguration. Statistical analysis using correlation techniques explore the dependency of service parameters on the resource utilisation.

### 2.1.5 Ease of Use.

The Network Monitor should be easy to use and measurement results should be displayed and reported in easily readable form. The following are expected to be supported[KP 87]:

Menu - driven techniques.

Selection of measurement.

Help facility.

Screen dump.

## 2.2 Design of Network Monitor for a Token - Ring Network.

With the criteria explained in previous section as a general guideline, a Network Monitor is developed for a Token Ring network the details of which are given in Appendix I. The monitor uses the same network interface with a different fimware.

A layered approach is employed in the development of Network Monitor software. It consists of three layers as described below:

### 2.2.1 Data Acquisition Layer.

This layer consists of the software that monitors the communication lines of the system for interpreting the network traffic. The programs for traffic interpretation are stored in PROMS and are protocol dependent. Once events of importance have occured, status or alert messages are prepared and passed to the Data Processing Layer, in a circular buffer , and in a standard format. Thus the low level protocol is transparent to Data Processing Layer.

### 2.2.2 Data Processing Layer.

The status messages in the circular buffer form the inputs to Data Processing Layer. It computes the various parameters regularly and updates the data bases at regular intervals of time. The results of computation are passed to the User Interface Layer in a set of variables. Thus the Data Processing Layer is transparent to the User Interface Layer.

### 2.2.3 User Interface Layer.

The User Interface Layer generates the various display screens and processes commands from the user. It uses the data passed on from the Data Processing Layer to refresh the screen at regular intervals.

The layered approach suggested above implies a modularity in the software. The next three chapters give the implementation details of the Network Monitor.

# CHAPTER 3

# DATA ACQUISITION LAYER

This chapter describes the software referred to as Data Acquisition Layer in section 2.2.1,that receives each packet circulated on the Token -Ring for interpreting the network traffic .

## 3.1 Functions of Data Acquisition Layer.

The PC - LAN card developed at IIT Kanpur with a modified firmware is used for the data acquisition module. In discussion that follows the data acquisition module will be referred to as Interface Card . The Intel 8088 processor, Intel 8256 MUART, EPROM and RAM mounted on the Interface Card constitute the digital interface to the network. The details of the PC - LAN developed at IIT Kanpur are given in Appendix I. A 'T' - Connection to the RXD line of the primary node enables the Interface Card to receive data flowing on the network.

Data that is being transmitted from another computer or device over the network is acquired over a serial line by the MUART. The software stored in PROM assumes the Token - Ring protocol of the PC - LAN. Once events of importance have occurred , for example arrival of

a packet, status messages are prepared and transmitted to the Network Monitor.

### 3.2 Implementation of On - Line systems under MS - DOS.

In the present project, a IBM PC/XT compatible with MS - DOS version 3.20 is used as a Network Monitor. MS -DOS imposes certain restrictions in the development of software for data acquisition. It is therefore important to explore the support provided by MS - DOS in this connection. This section discusses this aspect in detail.

As of version 3.20, MS - DOS is not a multitasking operating system. MS - DOS is not re-entrant, which means interrupt routines cannot call MS - DOS without the risk of 'blowing' the system away [JK 86]. This means in order to implement on-line systems missing portions required by the application need to be provided.

The simplest type of on-line program is one that does not require any multitasking and does all of its input and output through standard MS - DOS devices. Unfortunately, nothing much can be accomplished under these restrictions. It is necessary to extend MS - DOS in order to implement more advanced systems.

One of the many multitasking features that MS - DOS lacks is pre-emptive scheduling, which is the ability of the operating system to interrupt a currently running routine and start another one [JK 86]. Pre-emptive

scheduling cannot be added to MS - DOS, because it is not re-entrant. In order to overcome this limitation the following strategy is to be adopted.

a) First, divide application into separate tasks.

b) Then, organize the tasks into two groups: those that have critical timing constraints and those that do not.

c) Determine what types of operations must be performed in critical sections and also determine whether they can be accomplished without calling MS - DOS. This is because MS - DOS does not guarantee that it returns from a call in a set amount of time. Critical sections of programs must therefore bypass MS - DOS altogether even if it means sacrificing compatibility.

d) Finally, decide whether the program will be interrupted by outside events.

When interfacing to Data Acquisition Module an MS - DOS device driver can be used to acquire data. This would result in a device driver that is portable to other MS - DOS systems. If MS - DOS device driver is not used in the interests of modular programming , a separate routine must handle data acquisition. This routine, as part of MS - DOS or not, is the device driver. The device driver is usually interrupt driven and does the function of storing data in the buffer. The calling program reads the buffer to

obtain the buffered data.

### 3.3 Implementation details of Data Acquisition Layer.

The Data Acquisition Layer comprises of the following:

(1) Firmware on the Interface Card which sends status messages to the Network Monitor.

(2) Software in the Monitor PC which receives the status messages and buffers them in a circular buffer.

#### 3.3.1 Firmware on the Interface Card.

Certain conventions are followed in the status messages sent from the Interface Card. They are as follows:

(1) Whenever a packet is received by the Interface Card without error, a status message of four bytes is to be generated, whose format is given below:

| | | | |
|---|---|---|---|
| 02 | Source Address | Destination Address | Length of Packet |

(2) Whenever a packet is received by the Interface Card with error, the status message generated is to be of the following format.

| | | | |
|---|---|---|---|
| 09 | Source Address | Destination Address | Length of packet |

The algorithm for the firmware on the Interface Card the is given in Appendix II. The salient features of the firmware are explained below:

(1) Whenever a byte is received by the Interface Card, the MUART generates an interrupt to Intel 8088 processor.

(2) The interrupt service routine(RXINT) is executed that receives the byte into a buffer.

(3) The master loop is the routine(MASTER) that is continuously executed by the processor on the Interface Card. The master loop checks, whether a packet has been received whose status message is to be generated, in which case calls a procedure(PCOUTP) that transfers the four bytes constituting the status message into four buffer variables namely STAT_FIRST, STAT_SOURCE, STAT_DEST, STAT_LEN. If the packet length (in bytes) minus the header length (in bytes) is equal to the value specified in the length field, the value 02 is assigned to STAT_FIRST. If this is not true then, the packet received is in error and a value 09 is assigned to STAT_FIRST. It then raises an interrupt to the PC by raising the IRQ2 line high so that status message may be received by the Network Monitor PC.

### 3.3.2 Software in Monitor PC.

When the interrupt is raised by the Interface Card, the interrupt service routine reads the bytes of the status message from the Interface card into a circular buffer. A pointer BUFF_PTR points to the last byte received by the Network Monitor. The Data Processing Layer determines that a status message is ready to be read from the circular buffer

by comparing BUFF_PTR with another pointer(PIPE_PTR) . The PIPE_PTR points to the last byte read from the circular buffer. The circular buffer is implemented as an array of 32 Kbytes. The device driver is coded in Intel 8088 assembly language and is given in APPENDIX III.

The next chapter gives details of functions and implementation aspects of Data Processing Layer.

# CHAPTER 4

## DATA PROCESSING LAYER

This chapter discusses the software referred to as Data Processing Layer in Section 2.2.2, that processes the status messages buffered in the circular buffer, computes parameters and updates the Data Bases . The various parameters computed can be classified into three categories:

(1) Network Summary Statistics.

(2) Node List Statistics.

(3) Error Statistics.

A Data Base file is maintained for each of the categories listed above. The following sections define the parameters computed under each of the categories and explain the algorithm for the computation of the same.

### 4.1 Network Summary Statistics.

The parameters that are classified under this category are as follows:

(1) Throughput: The number of bytes circulated on the ring in a given interval of time is defined as Throughput. It is computed for the current minute once every second. Throughput in previous Minute, the average value & peak

value is computed at the end of every minute.

(2) Utilisation: The ratio of the number of bytes circulated on the ring in a given interval of time to the maximum number that can be circulated in the same interval of time (i.e. maximum possible Throughput) is defined as Utilisation. Utilisation in previous minute and the peak value is computed at the end of every minute.

$$\text{Utilisation in previous minute} = \frac{\text{Throughput in previous minute(in bytes)}}{\text{Maximum possible Throughput(in bytes )}}$$

(3) Packet Summary: The number of packets circulated on the ring is computed every second. At the end of every minute, the number of packets circulated in the previous minute, their average value & peak value is computed.

(4) Data Base Summary: The Data Base Summary lists the number of records in each of the Data Bases.

The Data Processing Layer includes software that determines the elapsed time and updates data buffers at regular intervals of time. This is accomplished by the use of Timer - Tick interrupt for executing time critical tasks. A brief discussion of the Timer - Tick interrupt is as follows:

The dynamic memories in a PC are refreshed by means of an hardware interrupt to the processor. In addition to refreshing the dynamic memories the interrupt service

routine also invokes a software interrupt(INT 1CH) called the Timer - Tick interrupt. Normally the Timer - Tick interrupt points to an instruction which causes the program to return from the interrupt service routine. But the user program may 'capture' this interrupt so that a portion of code may be executed at regular intervals of time.

When a status message is processed the value specified in the Length field of the status message, and the header length is added to the variable NO_OF_BYTES_SEC, that keeps a count of the Throughput in current minute. The format of the status message is explained in Section 3.3.1. Likewise, the variable NO_OF_PACKETS_SEC keeps a count of the number of packets circulated on the ring in one second. At the end of every second the Interrupt Service Routine of the Timer_Tick(INT 1CH), transfers the contents of NO_OF_BYTES_SEC and NO_OF_PACKETS_SEC into intermediate buffers THRPUT_INTM_BUF and PAKSUM_INTM_BUF respectively, each of which is an array that can store sixty integers. In other words, the values of NO_OF_BYTES_SEC and NO_OF_PACKETS_SEC of successive seconds are stored in consecutive positions in the respective arrays. This process of accumulation in the buffers is time critical, i.e. it is to be executed at the end of every second. Hence, it is performed in the Interrupt Service routine of the Timer_Tick interrupt which is given in figure 4.1.

```
{ Interrupt Service routine of INT1Ch that gets
                          executed every 55.54msec.}


procedure Int1CHandler(Flags,CS,IP,AX,BX,CX,
                              DX,SI,DI,DS,ES,BP:word);
interrupt;
begin
   Inc(SECCOUNT);
   if SECCOUNT=18 then
   begin
     SECCOUNT:=0;
     SECUPDATE:=1;
     RINGUPTIME:=RINGUPTIME + 1;
     THRPUT_INTM_BUF[INTM_BUF_PTR]:=NO_OF_BYTES_SEC;
     PAKSUM_INTM_BUF[INTM_BUF_PTR]:=NO_OF_PACKETS_SEC;
     INTM_BUF_PTR:=INTM_BUF_PTR + 1;
     NO_OF_PACKETS_SEC:=0;
     NO_OF_BYTES_SEC:=0;
     Inc(MINCOUNT);
     if MINCOUNT = 60 then
     begin
          MINCOUNT:=0;
          MINUPDATE:=1;
          INTM_BUF_PTR:=1;
     end;
  end;
end;
```

**Figure 4.1**

The contents of the intermediate buffers THRPUT_INTM_BUF and PAKSUM_INTM_BUF are accumulated in THRPUT_CUR and PAKSUM_CUR respectively at regular intervals. These variables are used by the User - Interface layer to display the Throughput and number of packets in current minute every second. The average value of Throughput and Number of Packets circulated on the ring is computed as follows:

$$\text{Average value of Throughput in previous minute (in bytes per second)} = \frac{\text{Throughput in Previous Minute}}{60}$$

$$\text{Average value of number of packets transmitted on the ring in previous minute (in packets per second)} = \frac{\text{Number of packets circulated on ring in previous minute}}{60}$$

The Data Base file corresponding to Network Summary Statistics is DBAS.DAT. At the end of every minute the Throughput , Utilisation and the Number of Packets on the ring in the previous minute is stored in the Data Base file with date and time stamping.

The parameters of Data Base Summary are as follows:

(1) Sample Time: The interval of time between two updates of Data Base is defined as Sample Time. The Sample Time is one minute as the Data Base is updated every minute.

(2) Number of records in DBAS.DAT.

DBAS.DAT is the Data Base file for Network Summary Statistics.

(3) Number of records in NLDBAS.DAT.

NLDBAS.DAT is the Data Base file for Node List Statistics.

(4) Number of records in ESDBAS.DAT.

ESDBAS.DAT is the Data Base file for Error Statistics.

## 4.2 Node List Statistics.

For each connection node on the network that is involved in communication, the following parameters are computed at the end of every minute:

(1) Number of packets transmitted by the node.

(2) Number of bytes transmitted by the node.

(3) Average packet size of the transmitted packets.

(4) Number of packets received by the node.

(5) Number of bytes received by the node.

A Data Base file NLDBAS.DAT is maintained to record the Node List Statistics. At the end of every minute, the parameters listed above are recorded for every node that was involved in communication in previous minute , with date and time stamping.

The algorithm to compute the above parameters and the data structures used for this purpose are explained below and the program listing is given in Appendix III:

(1) The PC - LAN developed at IIT Kanpur can support upto

255 nodes. Refer Appendix I for further details.

(2) Two arrays NODE_LIST_ARRAY1 and NODE_LIST_ARRAY2 each of size 1550 integers is defined. These arrays store the dynamic values of each of the parameters listed above.

(3) For the duration of one minute, one of the arrays (say NODE_LIST_ARRAY1) is used , and for the next minute the other array(say NODE_LIST_PTR2) is used. The array used in a particular minute depends on the value of the variable NODE_LIST_PTR. If the NODE_LIST_PTR has a value '1' in a given minute then NODE_LIST_ARRAY1 is used ,else NODE_LIST_ARRAY2 is used.

(4) Each connection node is allotted five successive bytes in each of the arrays in the ascending order of node addresses. The Nth byte of the allotted bytes may be indexed by an offset given by N + (Connection Node Address - 1) * 5 . The first byte of the allotted byte if non - zero, signifies that the particular node has been involved in communication and will be referred to as 'DIRTY' byte. It is assigned a value '1' if the given node is involved in transmission and a value '3' if the given node is only involved in receiving packets from a source node. The second byte of the allotted bytes stores the dynamic value of the number of bytes transmitted by the given node and will be referred to as COUNT_BYTE_TXMT byte. The third byte stores the number of packets received and will be referred to as

COUNT_PAK_RECV. The fourth and the fifth bytes keep a count of the number of packets transmitted and number of bytes received and will be referred to as COUNT_PAK_TXMT and COUNT_BYTE_RECV respectively as follows:

| | | | | |
|---|---|---|---|---|
| DIRTY BYTE | COUNT_BYTE _TXMT | COUNT_PAK _RECV | COUNT_PAK _TXMT | COUNT_BYTE _RECV |

(5) When no time critical tasks need to be executed, a series of computations are performed. One of which as explained earlier, is to compute the Throughput and number of packets in the current minute; the other is to process the status messages placed in the circular buffer. These status messages are used to compute the Node List Statistics and Error Statistics.

As explained in previous chapter when a packet circulates on the ring without errors, a status message is generated by the Data Acquisition layer in the format given below:

| | | | |
|---|---|---|---|
| 02 | Source Address | Destination Address | Length |

If a packet circulates on the ring with errors the format of the status message generated is as follows:

| | | | |
|---|---|---|---|
| 09 | Source Address | Destination Address | Length |

(6) The first byte of the status message indicates whether it represents a status message for a packet that circulated on the ring with error or without error. If the status message represents a packet without error then the computations of Node List Statistics are performed. The Source address byte of the status message is used to index the Node List array pointed to by the NODE_LIST_PTR to access the DIRTY byte of allotted five bytes. A value of '1' is assigned to the DIRTY byte. To the COUNT_BYTE_TXMT field is added the Length field of the status message. The value in COUNT_PAK_TXMT is incremented by one. The Destination address field in the status message is used to index the Destination nodes DIRTY byte. If this byte is not equal to '1', it is assigned a value '3'. The COUNT_PAK_RECV corresponding to the destination node is incremented by '1', while the Length field of the status message is added to COUNT_BYTE_RECV field.

(7) At the end of every minute, the NODE_LIST_PTR is assigned a value '2' if it was previously '1' and vice versa. On demand , the User - Interface layer processes the Node List array used in the previous minute and displays the various parameters for each connection node.

(8) The Data Base file NLDBAS.DAT is updated at the end of every minute with date and time stamping. The number of records added to the Data Base would depend on the number of

connection nodes involved in communication.

## 4.3 Error Statistics.

When a packet circulating on the ring is found to be in error, a status message is generated whose first byte is '09'. When such a status message is processed, a record is added to the Data Base file (ESDBAS.DAT) with the Source address, Destination address and the Length of the packet as the fields.

The justification in direct transfer of such information to the Data Base is in the fact that errors are inherently random and bursty. The User - Interface layer reads the records from the Data Base file on demand from the user.

The next chapter explains in detail the User Interface layer.

# CHAPTER 5

# USER INTERFACE LAYER

This chapter explains the software referred to as User Interface Layer in Section 2.2.3, that generates various network status displays, updates the screen data at regular intervals and processes commands from user.

## 5.1 Design Criteria for Network Status Displays.

The User Interface Layer is crucial in the design of Network Monitor especially when a IBM PC with MS - DOS is used for this purpose. This is because each of the following functions need to be executed continuously:

(1) An interrupt service routine should receive all the bytes of status messages generated by Interface Card as explained in Chapter 4.

(2) The contents of the circular buffer have to be processed every second as explained in Chapter 4.

(3) Screen data of certain parameters have to be refreshed every second.

(4) Commands from user, if any, have to be processed.

(5) Entire screen contents have to be updated every minute.

(6) Each of the three Data Bases has to be updated every minute.

In addition to the functions mentioned above, a user friendly interface is to be provided. The principal objective is not to overload the Network Monitor user with a large amount of information within short time frames. [KP 87] suggests the following options in a user interface:

(a) Overview - type starting screens.

(b) Menu - driven technique for inexperienced users.

(c) Meaningful use of function keys for help and status displays.

(d) Hardcopy option.

(e) Zooming on events under consideration.

(f) Powerful window management.

(g) Screen store and recall capability.

(h) Direct inquires for experienced users (e.g. variable dump).

It requires, therefore, that the screen management routines are fast and efficient so that they do not hinder the functions of Data Acquisition and Data Processing Layers and also perform the various functions of User Interface layer.

## 5.2 Design of User Interface layer.

The important features of the User Interface layer software are as follows:

(1) The different display screens are stored in RAM before

the Data Acquisition Layer is activated by enabling the interrupts. One by one, each of the screens are drawn and stored in the RAM. This fulfills the requirement of overview - type starting screens. Later, whenever the user requests a particular screen to be displayed, that screen is transferred from RAM to display. This method ensures a very fast generation of a display screen.

(2) The different display screens generated are

(a) Network Summary Statistics.

(b) Node List Statistics.

(c) Error Statistics.

(d) Graphic display for Throughput.

(e) Graphic display for Utilisation.

(f) Help screens for Network Summary Statistics, Node List Statistics & Error Statistics.

(g) Overview Help Screen.

The screen prints of the display screens are given in figures 5.1 to 5.3.

(3) Each of the display screens can be called by the press of a function key. This provides a meaningful use of function keys and also reduces the workload of the user.

(4) In the lower portion of each display a menu is provided.

(5) A variable dump is provided for experienced users which displays the current status of the important parameters in one display screen.

(6) Graphic displays are provided for Throughput and Packet Summary each of which provides a 'snapshot' of the respective parameters in last 60 seconds.

(7) There are two methods to save the screen content at any given instant:

(a) Saving the screen content in a file in the current drive and directory.

(b) Direct printing of the screen content on a graphic printer connected through LPT1 parallel printer port.

In saving the screen content in a file, the software prompts the user to type in the name of the file in which the screen content has to be saved. This is the quickest method to save the screen content. The saved screen may be displayed whenever required and may be even printed out later if necessary.

(8) An option to transfer program control to the operating system shell is also provided. The user may use this option to transfer the Data Bases from RAM disk to the hard

disk/floppy diskette or run any other program whose size is less than 40K Bytes. It is however required that control is transferred back to the monitor software within one minute, so that no data is lost or in error.

The algorithm for the User Interface Layer software is as follows:

(1) Each of the display screen is assigned a value. Whenever the user requests for a particular display screen, the corresponding value is stored in the variable FUNCT_KEY before transferring the screen from RAM to display.

(2) The User Interface layer software accesses the computed results of the Data Processing layer and updates the screen values once every second or once every minute as required. For example, the Throughput and number of packets in current minute in Network Summary Statistics display are updated once every second while the rest of the parameters are updated once every minute.

(3) Node List Statistics are displayed in the following manner:

The Node List Array used in the previous minute is determined by the content of NODE_LIST_PTR. The DIRTY byte, of the allotted five bytes in the Node List Array as explained in Chapter 4 for each connection node, is checked for a value of '1' or '3', in which case that node was involved in communication in the previous minute. The values

stored in the other four bytes are used to update the display.

(4) Whenever the user requests for Error Statistics to be displayed the contents of Data Base file ESDBAS.DAT are used to display the parameters.

(5) The Timer - Tick interrupt(INT 1CH) is used to keep track of the time elapsed . At the end of every second, the current date and time values are determined and displayed on each of the screens.

(6) When the user requests for a graphic display of Throughput or Packet curve, the data values of the previous sixty seconds, obtained from the intermediate buffers THRPUT_INTM_BUF and PAKSUM_INTM_BUF are used to draw the graph.

The next chapter discusses the possible enhancements to the work presented and also the conclusions drawn from the project.

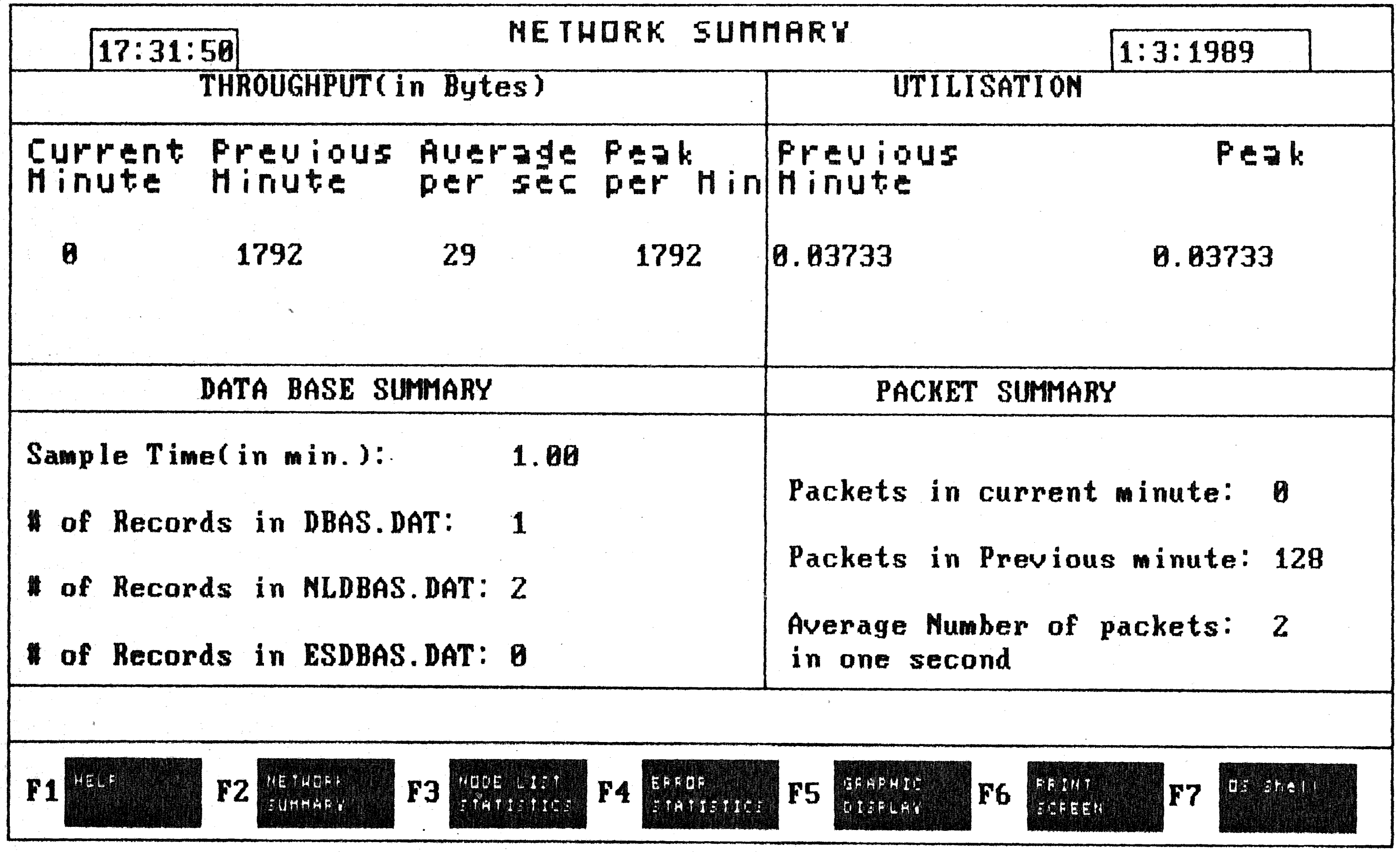

NETWORK SUMMARY
17:31:50
1:3:1989
THROUGHPUT(in Bytes)
UTILISATION
Current Minute
Previous Minute
Average per sec
Peak per Min
Previous Minute
Peak
0
1792
29
1792
0.03733
0.03733
DATA BASE SUMMARY
PACKET SUMMARY
Sample Time(in min.): 1.00
# of Records in DBAS.DAT: 1
# of Records in NLDBAS.DAT: 2
# of Records in ESDBAS.DAT: 0
Packets in current minute: 0
Packets in Previous minute: 128
Average Number of packets: 2
in one second
F1 HELP
F2 NETWORK SUMMARY
F3 NODE LIST STATISTICS
F4 ERROR STATISTICS
F5 GRAPHIC DISPLAY
F6 PRINT SCREEN
F7 OS Shell

Figure 5.1

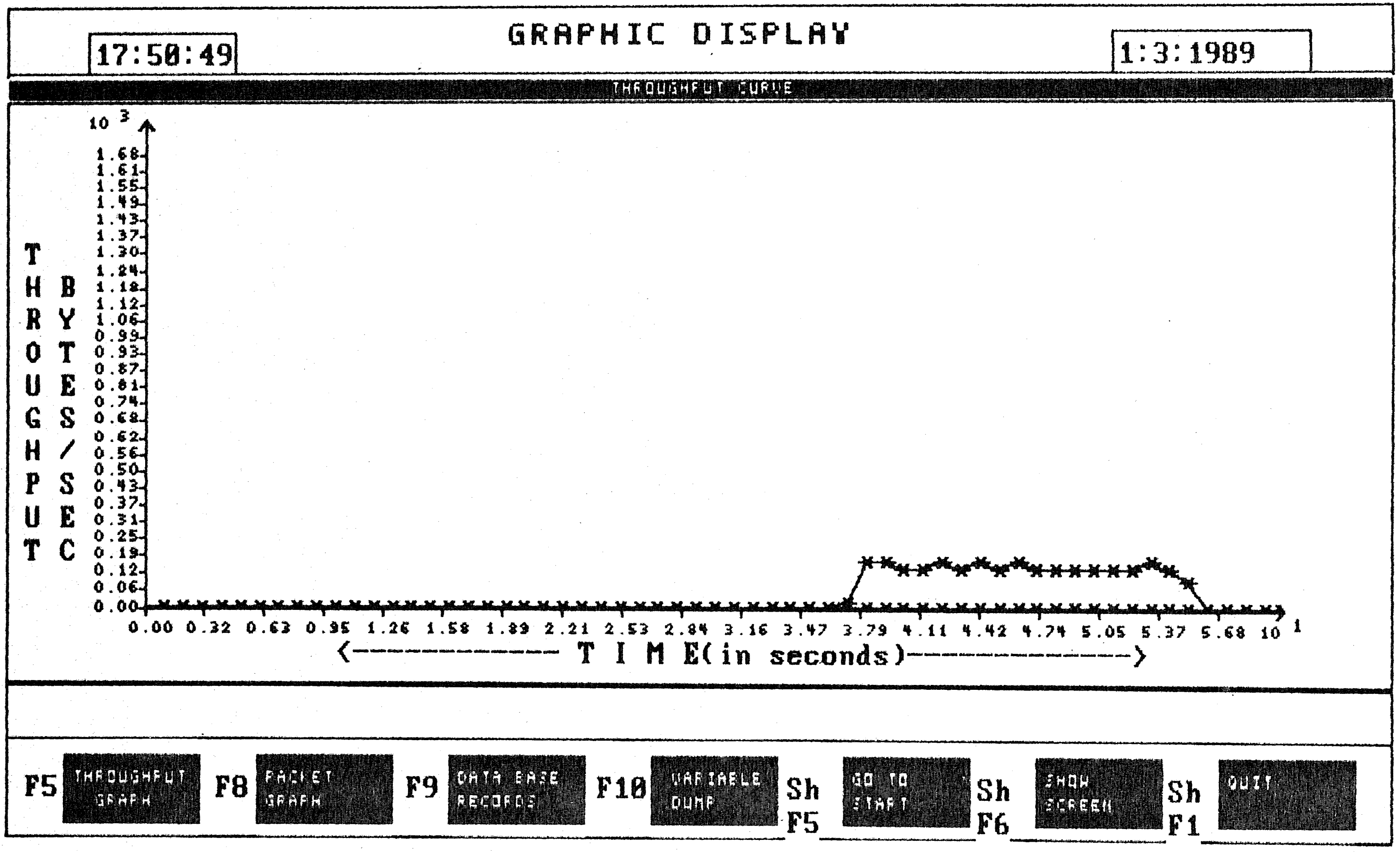
NODE LIST STATISTICS

17:48:21 1:3:1989

| Connection Node | Bytes Xmitted in previous minute | Pkts. Xmitted in previous minute | Average Pkt. Size | Pkts. rcvd. in previous minute | Bytes rcvd. in previous minute |
|---|---|---|---|---|---|
| 4 | 768 | 128 | 6 | 0 | 0 |
| 6 | 0 | 0 | 0 | 128 | 768 |

F1 HELP F2 NETWORK SUMMARY F3 NODE LIST STATISTICS F4 ERROR STATISTICS F5 GRAPHIC DISPLAY F6 PRINT SCREEN F7 OS SHELL

Figure 5.2



GRAPHIC DISPLAY
17:50:49
1:3:1989
THROUGHPUT CURVE
THROUGHPUT BYTES/SEC
10 3
1.68
1.61
1.55
1.49
1.43
1.37
1.30
1.24
1.18
1.12
1.06
0.99
0.93
0.87
0.81
0.74
0.68
0.62
0.56
0.50
0.43
0.37
0.31
0.25
0.19
0.12
0.06
0.00
0.00 0.32 0.63 0.95 1.26 1.58 1.89 2.21 2.53 2.84 3.16 3.47 3.79 4.11 4.42 4.74 5.05 5.37 5.68 10 1
T I M E(in seconds)
F5 THROUGHPUT GRAPH
F8 PACKET GRAPH
F9 DATA BASE RECORDS
F10 VARIABLE DUMP
Sh F5 GO TO START
Sh F6 SHOW SCREEN
Sh F1 QUIT


Figure 5.3

# CHAPTER 6

# CONCLUSIONS

This chapter describes the result of the implementation and gives suggestions for further improvement in performance.

A IBM PC compatible with MS - DOS version 3.3 and with a 4.77 Mhz clock was used as a Network Monitor and the performance was tested on the PC - LAN developed at IIT Kanpur with two nodes. The network activity was measured with the help of the Monitor, the sample screen prints of which are given in figures 5.1 to 5.3.

The Data Acquisition Layer was implemented in assembly language and the Data Processing Layer was implemented in Turbo Pascal. The Turbo Graphix Tool Box was used in the generating the various display screens of the User Interface Layer and Turbo Data Base Tool Box was used in the development of the Data Bases. This approach considerably reduced the system development time without sacrificing efficiency or flexibility.

The Network Monitor could be used to monitor any other network if the Data Acquisition Layer is suitably modified so that status messages are sent to the Monitor in the format as discussed in Chapter 3.

The Network Monitor has been tested for the required performance for data rates of 192 Kbps. It could be used for higher data rates if a IBM PC/AT or a PC with higher clock speed is used.

The design has the flexibility to include measurement of parameters other than those already discussed in this project. Thus it is possible to tailor the software to the specific requirement of the user.

With the above mentioned features, we hope that the Network Monitor would give a satisfactory performance to the user.

# APPENDIX I

## Details of Implementation of PC - LAN at IIT Kanpur

The following description gives the software and hardware details of the PC - LAN at IIT Kanpur[SV 88].

### A .1. OVERVIEW of the PC - LAN.

The PC - LAN is a ring network designed and implemented using Intel's 8256 MUART. Each node transmits packets only when it captures a token. An Interface Message Processor(IMP) card allows communication of the host PC with the ring. The data transfer process between the card and the Host is interrupt driven . The network is operated at 192 Kbps. One of the nodes on the ring is designated as 'primary', the main function of which is to initiate token at start-up time and when a token is lost. Each of the other nodes on the ring is designated as 'secondary'.

### A .2 Hardware Details of PC - LAN.

The hardware explained above consists of a IMP card interfaced with the PC bus. The card essentially consists of a Intel 8088 CPU, a Intel 8256 MUART, a clock recovery circuit and driver/receiver circuit. The 74LS138 decoder has been used to decode the address lines on the PC side. The PC-XT uses address ranges 340H - 347H and 360H -

360H - 367H have been used to read from the PC side. The address ranges 368H - 36FH to write any byte to the IMP card. RXD line of MUART is connected to preceding node's TXD , and TXD line is connected to succeeding nodes RXD through appropriate line driver and receiver. 75110 line driver and 75108 line receiver have been used to provide a balanced current drive. whenever the MUART has to transfer a byte to the PC ,it pulls the IRQ2 line high ,which causes an interrupt to the PC. The PC ,then reads the byte from the IMP card. The transmit portion of the MUART consists of a transmit buffer, a transmit register and the associated control logic. The data byte to be transmitted is first written to transmit buffer. The data transfer from the transmit register to transmit buffer takes place during the transmission of start bit.

A .3 **Software details of PC - LAN.**

The ISO reference model for Open System Interconnection has been followed in the development of communication software. The data communication software is implemented in the IMP card which communicates with the host (PC - XT) through an I/O port for further processing by the upper layers. The primary node software is similar to secondary node with some minor modifications. Primary node is the node which initiates the circulation of the token on the ring when power is first turned on. The removal of circulating packets

is also performed by the primary node. Further, the primary puts the received packet in repeat buffer, if the packet is not addressed to it, and retransmits when it gets back the token. In contrast to this, the secondary node repeats the packet received from the network. Apart from the above differences, the primary node software is similar to secondary node software.

Packet structure:-

The following packet structure format has been used for communication.

| STX | DEST | CONTROL | SOURCE | DISPOS | LEN1 | LEN2 | ..DATA.. | ETX |
|---|---|---|---|---|---|---|---|---|

For bytes between STX and ETX , if byte to be sent is same as STX or EOP ,then DLE - STX or DLE - EOP are respectively sent. The receiver deletes the stuffed DLE characters from the received data. The STX, ETX, EOP and DLE are standard ASCII characters as given below.

STX = 02H , ETX = 03H , EOP = 7FH , DLE = 10H

The DEST and SOURCE bytes give the destination and source address respectively. The primary address has been chosen to be equal to 00H and the addresses of other nodes may vary from 01H to 0FEH. The address 0FFH has been used for broadcast packets.

Three bits of CONTROL byte are currently being

used. C7 indicates whether the packet is going from primary to secondary or secondary to primary. C6 indicates the sequence number of packet and C5 is used to distinguish between an acknowledgment packet and an ordinary packet. DISPOS is an extended control byte, which is not currently used.

# APPENDIX II

## ALGORITHM OF THE DATA ACQUISITION LAYER

This appendix describes the various data structures used by the Data Acquisition Layer and also the algorithm of the various routines of the software.

### A2.1 Data Structures used by the Data Acquisition Layer:

**ROUTE** One byte in RAM, Two bits R3 and R1 are used. ROUTE reports the status of operations.

| | | |
|---|---|---|
| R3 | 1 | received packet waiting in PC, the status message of which has to be generated and sent to PC |
| | 0 | otherwise, nothing waiting to be output PC |
| R1 | 1 | node is currently receiving a packet |
| | 0 | otherwise |

**VECTOR** vector for RXINT of the 8256

**RXCHAR** RAM location containing the current byte received

**RXPRE** RAM location containing the previous byte received

**RCVBUF** Four buffers RCVBUF0-3, each of length (255 + 7) bytes, RCVBUF0...RCVBUF3 are also 2-byte constants indicating the start addresses of the respective buffers. These are used to store received packets. Each of these receive buffers is organised as follows -

| | |
|---|---|
| BUFSTAT | B7 - B0 status of this buffer |
| DEST | Destination address of received packet |
| CONTROL | Control field of received packet |
| SOURCE | Source address of received packet |
| DISPOS | Disposition byte |
| LEN1 | Equal to 00 |
| LEN2 | Length of packet(0 - 255) |

......

| | |
|---|---|
| data | Actual data content |
| ...... | |
| ETX | Tail Marker needed to detect error packets |

Initialise BUFSTAT of each RCVBUF to 00 at start-up

The individual bits B7- B0 of each BUFSTAT imply the following -

| | |
|---|---|
| B7 - B3 | unused |
| B2 = 1 | this buffer is being emptied to the PC |
| = 0 | otherwise |
| B1 = 1 | this buffer is being filled from network |
| = 0 | otherwise |
| B0 = 0 | this buffer is empty (or being filled) |
| = 1 | otherwise(being full or being |

**RCVINP** 2-byte pointer in RAM indicating the location in RCVBUF to be filled next with byte from network

**BUFPNTI** BI7-BI0 A 1-byte pointer in RAM indicating the RCVBUF to be used next to input from network(if it is empty;if it is not empty, data cannot be accepted from the network)

Initialise BUFPNTI to 00 at start up. The individual bits of BUFPNTI imply the following

| | | |
|---|---|---|
| BI7 - BI2 | | unused |
| BI1,BI0 | 00 | for RCVBUF0 |
| | 01 | for RCVBUF1 |
| | 10 | for RCVBUF2 |
| | 11 | for RCVBUF3 |

**STAT_FIRST** stores first byte of status message

**STAT_SOURCE** stores the source address field of the arrived packet and therefore of the status message

**STAT_DEST** stores the destination field of the arrived packet

**STAT_LEN** stores the length field of the arrived packet

Other RAM storage locations for internal processing

| | |
|---|---|
| DESTR | 1-byte |
| CNTRLR | 1-byte |
| SRCER | 1-byte |
| DISPOS | 1-byte |
| LENR1 | 1-byte |
| LENR2 | 1-byte |
| RCVIN | 2-byte back-up address pointer |

**A2.2 Algorithm of the firmware**

**SECOND** Initialisation routine for secondary

[1] Initialize 8256 for network interface.

[2] Initialize RXPRE = 00H
ROUTE = 00H
RXCHAR = 00H

[3] Initialize BUFSTAT = 00 for each RCVBUF (0 - 3)
BUFPNTI = BUFPNTO = 00H

[4] Initialize stack, enable RXINT and go to MASTER

**MASTER**

[1] if R3 of ROUTE is 1 then call PCOUTP
else go to [1]

**RXINT** Interrupt Service Routine executed on receiving a character

[1] Save all registers on stack
[2] (RXPRE) = (RXCHAR)
[3] (RXCHAR)= Rcvd. byte from 8256
[4] If Rcvd. Byte in error ---

```
    (a)   (RXCHAR)=00
          Reset 8256 error flags
          Jump to INIT


     Else if (RXCHAR)=STX or EOP
           (b)  Jump to instruction at location VECTOR
          else continue to [5]

[5] If (RXPRE)=DLE go to [4b]; else continue
[6] If (RXCHAR)=STX, then

    (a) Reset R7=R6=R5=R4=R1=0
    (b) For BUFPNTI = 00          RCVINP = RCVBUF0
            BUFPNTI = 01          RCVINP = RCVBUF1
            BUFPNTI = 10          RCVINP = RCVBUF2
            BUFPNTI = 11          RCVINP = RCVBUF3
    (c) RCVIN=RCVINP
    (d) VECTOR=STX_FOUND
    (e)  Restore  all registers (ie. those saved by  [1]  of
                                 RXINT)
    (g) Reenable interrupts
    (h) Return

[7] If (RXCHAR)=EOP, then ----

    (a) Reset R7=R6=R5=R4=R1=0
    (b) For BUFPNTI = 00          RCVINP = RCVBUF0
            BUFPNTI = 01          RCVINP = RCVBUF1
            BUFPNTI = 10          RCVINP = RCVBUF2
            BUFPNTI = 11          RCVINP = RCVBUF3
    (c) RCVIN=RCVINP
    (d) VECTOR=INIT
```

**INIT**

```
[1] Reset R7=R6=R5=R4=R1=0
[2] VECTOR=INIT
[3] Restore all registers (ie. those saved by [1] of RXINT)
[4] Reenable interrupts
[5] Return
```

**STX_FOUND**

```
[1] (DESTR)=(RXCHAR)  VECTOR=DEST_FOUND
[2] Restore all registers (ie. those saved by [1] of RXINT)
[3] Reenable interrupts
[4] Return
```

**DEST_FOUND**

```
[1] If (RXPRE)=DLE and {(RXCHAR)=STX or (RXCHAR)=EOP},then
                           go to [1] of STX_FOUND;
    Else continue.
[2] (CNTRLR)=(RXCHAR)    VECTOR=CNTRL_FOUND
[3] Restore all registers (ie. those saved by [1] of RXINT)
[4] Reenable interrupts
[5] Return
```

**CNTRL_FOUND**

```
[1] If (RXPRE)=DLE and {(RXCHAR)=STX or (RXCHAR)=EOP}, then
       (CNTRLR)=(RXCHAR) and go to [4] of DEST_FOUND;
    Else continue.
[2] (SRCER)=(RXCHAR)      VECTOR=SOURCE_FOUND
[3] If CNTRLR7=0 jump to INIT
    If CNTRLR7=1 then go to [4].
[4]  Restore  all  registers   (ie. those saved  by  [1]  of
                                    RXINT)
[5] Reenable interrupts
[6] Return
```

**SOURCE_FOUND**

```
[1] If (RXPRE)=DLE and {(RXCHAR)=STX or (RXCHAR)=EOP),then -
          (SRCER)=(RXCHAR) and go to [4] of CNTRL_FOUND;
    Else continue.
[2] (DISPOS)=(RXCHAR)       VECTOR=DISPOS_FOUND
[3] If CNTRLR7=1 and CNTRLR5=1 then R4=1
[4] Restore all registers  (ie. those saved by [1] of RXINT)
[5] Reenable interrupts
[6] Return
```

**DISPOS_FOUND**

```
[1] If (RXPRE)=DLE and {(RXCHAR)=STX or (RXCHAR)=EOP}, then
        (DISPOS)=(RXCHAR)  and  go to [5]  of  SOUND_FOUND;
     Else continue.
[2] (LENR1)=(RXCHAR)     VECTOR=LEN1_FOUND
[3] Restore all registers     (ie.  those saved by [1] of
                                      RXINT)
[4] Reenable interrupts
[5] Return
```

**LEN1_FOUND**

```
[1] If (RXPRE)=DLE and {(RXCHAR)=STX or (RXCHAR)=EOP}, then
       (LENR1)=(RXCHAR) and go to [3] of DISPOS_FOUND;
    Else continue.

[2] (LENR2)=(RXCHAR)          VECTOR=DATA_FIND
[3] If (LENR1) = 00           jump to INIT
[4]  Restore all registers    (ie. those saved by [1] of
                                           RXINT)
[5] Reenable interrupts
[6] Return
```

**DATA_FIND**

```
[1] If (RXPRE)=DLE and {(RXCHAR)=STX or (RXCHAR)=EOP} then
       (LENR2)=(RXCHAR) and go to [4] of LEN1_FOUND;
    Else continue.
[2] (COUNT)=(LENR2)
[3] If R4=0 then

     (RCVINP) = xxxxx010     ie. B2,B1,B0  Set R1=1
     RCVINP=RCVINP+1
     (RCVINP)=(DESTR)         RCVINP=RCVINP+1
     (RCVINP)=(CNTRLR)        RCVINP=RCVINP+1
     (RCVINP)=(SRCER)         RCVINP=RCVINP+1
     (RCVINP)=(DISPOSR)       RCVINP=RCVINP+1
     (RCVINP)=(LENR2)         RCVINP=RCVINP+1

[4] If (COUNT)=00 go to SEARCH_ETX
[5] If R4=0 then
      {a}    (RCVINP)=(RXCHAR)
             RCVINP=RCVINP+1
             go to [6]

[6] (COUNT)=(COUNT)-1

    If (COUNT)=00 then VECTOR=SEARCH_ETX
    Else VECTOR=DATA_FOUND

[7]  Restore all registers    (ie. those saved by [1] of
                              RXINT)

[8] Reenable interrupts
[9] Return
```

**DATA_FOUND**

[1] If (RXPRE)=DLE and {(RXCHAR)=STX or (RXCHAR)=EOP} then
 [a] RCVINP=RCVINP-1 COUNT=COUNT+1
 Else continue.
[2] If R4=0 go to [5a] of DATA_FIND
[3] Go to [6] of DATA_FIND

**SEARCH_ETX**

[1] If (RXPRE)=DLE and {(RXCHAR)=STX or (RXCHAR)=EOP} then
 go to [1a] of DATA_FOUND
[2] If (RXCHAR)=ETX, jump to INIT
[3] If R4=0 then
 {a} (RCVIN)=xxxxx001 (mark buffer full)
 {b} Increment BUFPNTI by 1 (mod 4 increment)
 {c} Set R3=1

[4] Jump to INIT

**PCOUTP**

[1] Save registers
[2] Disable interrupts
[3] If buffer full then transfer values into STAT_SOURCE, STAT_DEST, STAT_LEN. Check for Tail Marker at the end of packet. If found then STAT_FIRST = 02 else packet received is in error and STAT_FIRST=09
[4] Transfer the bytes STAT_FIRST, STAT_SOURCE, STAT_DEST, STAT_LEN to port
[5] Enable interrupts
[6] Restore registers
[7] Return

## APPENDIX III

## PROGRAM LISTING FOR DATA PROCESSING AND USER INTERFACE LAYERS

```
; The following is the assembly language program listing of
; the source code that receives the bytes of the status
; messages and places them in circular buffer. This program
; is assembled to a file HEADERAQ.OBJ and linked to main
; program coded in pascal.

Name Headeraq.asm

data segment  PUBLIC
EXTRN   BYTEFLAG:byte,ERROR:byte,UPDATE:byte
EXTRN   BUFF_PTR:word,RINGBUFF:byte
EXTRN   NEW_PKT_FLAG:byte,MINUPDATE:word
EXTRN   NO_OF_BYTES_SEC:word,ASCII_VALUE:byte,NO_KEY:word
EXTRN   PAKSUMCUR_MIN:word,NO_OF_PACKETS_SEC:word
EXTRN   THRPUT_INTM_BUF:word,PAKSUM_INTM_BUF:word,RESULT:byte
EXTRN   INTM_BUF_PTR:word,RINGUPTIME:word
EXTRN   SECCOUNT:word,MINCOUNT:word,SECUPDATE:word
data ends
code segment PUBLIC
     assume cs:code,ds:data,es:data
     PUBLIC INITIRQ2
     PUBLIC IRQ2_Handler

; The following is the interrupt service routine executed
; when the Interface Card raises a IRQ2 interrupt .

IRQ2_Handler proc near;Routine A1
      sti
      push ax
      push si
      push dx
      push ds
      mov ax,data
      mov ds,ax
      mov dx,340h
      in  al,dx
      inc BYTEFLAG
      cmp BYTEFLAG,04
      jnz last
      mov BYTEFLAG,00
      add NO_OF_BYTES_SEC,08h
```

```
      mov ah,00
      add NO_OF_BYTES_SEC,ax
      inc NO_OF_PACKETS_SEC
 last:cli
      mov si,BUFF_PTR
      mov RINGBUFF[si],al
      inc BUFF_PTR
      cmp BUFF_PTR,32000
      jnz cont
      mov BUFF_PTR,0000
cont: sti
      mov al,20h
      out 20h,al
      pop ds
      pop dx
      pop si
      pop ax
      iret
 IRQ2_Handler endp


; The following is the routine that enables the IRQ2
; interrupt and invoked by a main program coded in
; Pascal(NETWORK MONITOR)

 INITIRQ2 proc near;Routine A2

 ;------Set up the communication interrupt vector for IRQ2
       push ds
       push dx
       push ax
       mov dx,OFFSET IRQ2_Handler
       mov ax,SEG IRQ2_Handler
       mov ds,ax
       mov al,0ah
       mov ah,25h
       int 21h
  ;--------Clear the interrupt mask register of 8259
       mov al,00000000b
       out 21h,al
       pop ax
       pop dx
       pop ds
       ret
  INITIRQ2 endp
  code ends
  end
```

```
{ The following is the type declaration of the records to be
stored  in each of the Data Bases }

type
   key1=string[6];
   key2=string[8];
   NETDBSREC=record
       RecStatus:longint;
       DBSTIME:key1;
       DBSDATE:key2;
       TPUT:longint;
       PKET:longint;
       end;
  NODELISTDBSREC = record
       RecStatus:longint;
       DBSTIME:key1;
       DBSDATE:key2;
       NODEAD:integer;
       NUMBERPAK:longint;
       PAKRCV:longint;
       NUMBERBYTE:longint;
       AVERAGESIZE:integer;
       RECEIVEDNO:longint;
       end;
   ERRORSTATDBSREC = record
       RecStatus:longint;
       DBSTIME:key1;
       DBSDATE:key2;
       SOURCE_ADDRESS:integer;
       DEST_ADDRESS:integer;
       LENGTH_PACKET:longint;
       end;
  MaxDataType=NODELISTDBSREC;
  MaxKeyType=key2;

PROGRAM NETWORK MONITOR;
{$L HEADERAQ}
{$M $4000,250000,350000}
uses Crt,Dos,Turbo3,Graph3,Graph,Printer,Gdriver,Gwindow,
     Gshell,Gkernel,displib,taccess,tahigh;
{$I DATABASE.TYP}

{ USERINT is the compiled library of routines that generates
the  various display screens, while taccess and  tahigh  are
the  units  of Turbo Data Base ToolBox generated  with  the
type declarations given DATABASE.TYP. The rest of the  units
specified  in  the uses statement are the Turbo  Pascal  and
Turbo Graphix Toolbox }
```

```
label  200,500;
const
     MAX_BYTE_IN_SEC=800;

type
    DBSTYPE=record
            RecStatus:longint;
            DBSTIME:string[6];
            DBSDATE:string[8];
            TPUT:longint;
            PKET:longint;
            end;
    NODELISTDBSTYPE=record
            RecStatus:longint;
            DBSTIME:string[6];
            DBSDATE:string[8];
            NODEAD:integer;
            NUMBERPAK:longint;
            PAKRCV:longint;
            NUMBERBYTE:longint;
            AVERAGESIZE:integer;
            RECEIVEDNO:longint;
            end;
     ERRORSTATDBSTYPE=record
            RecStatus:longint;
            DBSTIME:string[6];
            DBSDATE:string[8];
            SOURCE_ADDRESS:integer;
            DEST_ADDRESS:integer;
            LENGTH_PACKET:longint;
            end;

     VecPtr = Pointer;
var
  { variables for NETWORK  SUMMARY correspondingly
    iniatialised by INITNETSUM  procedure   }
                RINGUPTIME:integer;
                RINGDOWNTIME:integer;
                SAMPLETIME:real;
                THRPUTPEK_OVER_HR:longint;
                THRPUT_AVG_MN:longint;
                THRPUT_IN_PREV_MIN:longint;
                THRPUTCUR_MIN:longint;
                UTILIS_AVG_MN:real;
                UTILISPEK_OVER_HR:real;
                UTILIS_IN_PREV_MIN:real;
                PAKSUM_IN_PREV_MIN:longint;
                PAKSUM_AVG_MN:longint;
                PAKSUMCUR_MIN:longint;
                PAKSUMPEK_OVER_HR:longint;
```

```
                    NO_OF_MINUTES:integer;
                    MAX_BYTE_IN_MINUTE:longint;
                    RECORDS_IN_DBAS:integer;
                    RECORDS_IN_NLDBAS:integer;
                    RECORDS_IN_ESDBAS:integer;
 { variables for ASSEMBLY language routines correspondingly
   initialised by INITASSEMBLYVAR procedure   }
     SECCOUNT,MINCOUNT:word;
     SECUPDATE,MINUPDATE,FIRST,NO_OF_BYTES_SEC:integer;
     NO_OF_PACKETS_SEC:integer;
     BYTEFLAG,ERROR,UPDATE:byte;
     BUFF_PTR:integer;
     RINGBUFF:array[0..31999] of byte;
     NEW_PKT_FLAG:byte;
{ variables used for EVERY SECOND PROCESSING correspondingly
  initialised by INITPROCESSEC procedure }
                   UP_TP_PREV_SEC:real;
                   SKIP_FLAG:integer;
                   READ_BYTE:byte;
                   PIPE_FLAG:word;
                   PIPE_PTR:word;
                   MATCH_INTM_BUF_PTR:integer;
                   ALREADY_IN,NO_KEY:integer;
                   ASCII_VALUE:char;
                   SOURCEAD,DESTAD,HEADER_STATUS:integer;
                   KOUNT,LENGTH:integer;
 { variables used by GRAPHIC routines correspondingly
   initialised by INITGRAPHICVAR procedure  }
      THRPUT_OVER_HR:array [1..60] of longint;
      PAKSUM_OVER_HR:array [1..60] of longint;
      UTILIS_OVER_HR:array [1..60] of real;
      THRPUT_SEC:PlotArray;
      UTILIS_SEC:PlotArray;
      PAKSUM_SEC:PlotArray;
      THRPUT_SEC_PTR:integer;
      UTILIS_SEC_PTR:integer;
      PAKSUM_SEC_PTR:integer;
      THRPUT_MIN_PTR:integer;
      UTILIS_MIN_PTR:integer;
      PAKSUM_MIN_PTR:integer;
      THRPUT_INTM_BUF:array[1..60] of integer;
      INTM_BUF_PTR:integer;
      PAKSUM_INTM_BUF:array[1..60] of integer;

  { variables used by NODE LIST STATISTICS
    correspondingly initialised by INITNODELISTVAR
    procedure }
                NODE_LIST_ARRAY1:array[0..1550] of longint;
                NODE_LIST_ARRAY2:array[0..1550] of longint;
                NODE_LIST_PTR:integer;
```

```
            INDEX:integer;OFFSET_VALUE:integer;
            NODE_LIST_UPDATE_FLAG:integer;

  { variables  used by ERROR LIST STATISTICS
    correspondingly initialised by INITERRORLISTVAR
    procedure }
        PAKINERROR:longint;
        ERROR_FLAG:integer;
        CURRENT_ERROR_LIST_PTR:integer;
        NO_OF_TOKEN_LOSSES:integer;

  { MISCELLANEOUS and DATA BASE variables that need
    not be iniatialised  }

        GraphDriver,GraphMode,I:integer;
        FUNCT_KEY,PREV_FUNCT_KEY:integer;
        Int1CSave:VecPtr;fname_str:string[12];
        ch,chr:char;RESULT:byte;

        dbas,nldbas,esdbas:DataFile;
        NumberRec:longint;

        THROUGHPUT:longint;
        PACKET:longint;
        DDATE:string[8];
        DTIME:string[6];
        STHR:string[2];
        STMN:string[2];
        STSC:string[2];
        STYYYY:string[4];
        STMM:string[2];
        STDD:string[2];


        InputRec:DBSTYPE;
        DBSREC:DBSTYPE;
        NLDBSREC:NODELISTDBSTYPE;
        ESDBSREC:ERRORSTATDBSTYPE;
{ procedure to open  the data base file DBAS.DAT in
                                        virtual disk D }
   procedure  NetSum_DataBase_Build; {Routine #1}
    begin
     OpenFile(dbas,'D:DBAS.DAT',SizeOf(DBSREC));
     if not OK then
     MakeFile(dbas,'D:DBAS.DAT',SizeOf(DBSREC));
     if not OK then
     write('Could not open or create the data file');
    end;
{ procedure to open the data base file NLDBAS.DAT in
                                        virtual disk D}
```

```
  procedure NodeList_DataBase_Build;(Routine #2)
  begin
   OpenFile(nldbas,'D:NLDBAS.DAT',SizeOf(NODELISTDBSREC));
   if not OK then
   MakeFile(nldbas,'D:NLDBAS.DAT',SizeOf(NODELISTDBSREC));
   if not OK then
   write('Could not open or create the data file');
  end;
{ procedure to open the data base file ESDBAS.DAT in
                                          virtual disk D }
procedure Errorstat_DataBase_Build;(Routine #3)
  begin
   OpenFile(esdbas,'D:ESDBAS.DAT',SizeOf(ERRORSTATDBSREC));
   if not OK then
   MakeFile(esdbas,'D:ESDBAS.DAT',SizeOf(ERRORSTATDBSREC));
   if not OK then
   write('Could not open or create the data file');
  end;

{ procedure to  update NODE LIST STATISTICS data base    }
procedure NODELISTDBSUPDATE;(Routine #4)
    begin
         NLDBSREC.RecStatus:=00;
         Str(HR,STHR);
         Str(MN,STMN);
         Str(SC,STSC);
         NLDBSREC.DBSTIME:=Concat(STHR,':',STMN,':',STSC);
         Str(YYYY,STYYYY);
         Str(MM,STMM);
         Str(DD,STDD);
         NLDBSREC.DBSDATE:=Concat(STMM,':',STDD,':',STYYYY);
         AddRec(nldbas,NumberRec,NLDBSREC);
         RECORDS_IN_NLDBAS:=NumberRec;
    end;
{ procedure to update ERROR STATISTICS data base         }
 procedure ERRORSTATDBSUPDATE;(Routine #5)
    begin
         ESDBSREC.RecStatus:=0;
         Str(HR,STHR);
         Str(MN,STMN);
         Str(SC,STSC);
         ESDBSREC.DBSTIME:=Concat(STHR,':',STMN,':',STSC);
         Str(YYYY,STYYYY);
         Str(MM,STMM);
         Str(DD,STDD);
         ESDBSREC.DBSDATE:=Concat(STMM,':',STDD,':',STYYYY);
         AddRec(esdbas,NumberRec,ESDBSREC);
         RECORDS_IN_ESDBAS:=NumberRec;
    end;
```

```
{ procedure to display the records in DBAS.DAT    }
procedure DISPDBSFILE;{Routine #6}
var NumberOfRecords,RecordNumber:longint;KOUNT:integer;
begin
  NumberOfRecords:=FileLen(dbas);
  ClearScreen;
  GotoXY(1,1);
  writeln('Records in Network Summary Data Base');
  KOUNT:=0;
  for RecordNumber:=1 to NumberOfRecords-1 do
  begin
    Inc (KOUNT);
    GetRec(dbas,RecordNumber,DBSREC);
    if DBSREC.RecStatus=0 then
    begin
      with DBSREC do
          begin
            writeln;
            writeln('TIME:',DBSTIME,' DATE:',DBSDATE,
                  '  THROUGHPUT:',TPUT,'  PACKET:',PKET);
          end;
    end;
    if DBSREC.RecStatus=1 then
    begin
      writeln('InputRec RecStatus=1',
                              '  NumberRec=',RecordNumber);
    end;
    if (KOUNT=8) then begin
                  KOUNT:=0;
                  GotoXY(2,22);
                  write('PRESS RETURN TO CONTINUE...');
                  repeat
                    ch:=readkey;
                  until (ch=#13);
                  ClearScreen;
                  GotoXY(1,1);
                         end;
              end;
        end;
{ procedure which displays the records in NLDBAS.DAT      }
procedure DISPNLDBSFILE;{Routine #7}
var NumberOfRecords,RecordNumber:longint;KOUNT:integer;
begin
   NumberOfRecords:=FileLen(nldbas);
   ClearScreen;
   GotoXY(1,1);
   writeln('Records  in Node List Statistics Data Base');
   KOUNT:=0;
   for RecordNumber:=1 to NumberOfRecords-1 do
   begin
```

```
    Inc (KOUNT);
    GetRec(nldbas,RecordNumber,NLDBSREC);
    if NLDBSREC.RecStatus=0 then
    begin
        with NLDBSREC do
        begin
        writeln;
        writeln('TIME:=',DBSTIME,'DATE:=',DBSDATE,
        'NODEAD:=',NODEAD,'PAKXMIT:=',NUMBERPAK,
        'PAKRCV:=',PAKRCV,':',NUMBERBYTE,':',AVERAGESIZE);
    end;
  end;
  if NLDBSREC.RecStatus=1 then
  begin
     writeln('InputRec RecStatus=1',
                                 '  NumberRec=',RecordNumber);
  end;
  if (KOUNT=8) then begin
                  KOUNT:=0;
                  GotoXY(2,22);
                  write('PRESS RETURN TO CONTINUE...');
                  repeat
                    ch:=readkey;
                  until (ch=#13);
                  ClearScreen;
                  GotoXY(1,1);
                       end;
   end;
  GotoXY(2,22);
  write('PRESS RETURN TO CONTINUE.....');
  repeat
  ch:=readkey;
  until (ch=#13);

 end;
{ procedure which displays the records in ESDBAS.DAT  }
procedure DISPESDBSFILE;(Routine #8)
 var NumberOfRecords,RecordNumber:longint;KOUNT:integer;
 begin
      NumberOfRecords:=FileLen(esdbas);
      KOUNT:=0;
      for RecordNumber:=1 to NumberOfRecords-1 do
      begin
        Inc (KOUNT);
        GetRec(esdbas,RecordNumber,ESDBSREC);
        if ESDBSREC.RecStatus=0 then
        begin
          with ESDBSREC do
          begin
            GotoXY(2,KOUNT + 8);
```

```
          write(DBSDATE);
          GotoXY(12,KOUNT + 8);
          write(DBSTIME);
          GotoXY(24,KOUNT + 8);
          write(SOURCE_ADDRESS);
          GotoXY(41,KOUNT + 8);
          write(DEST_ADDRESS);
          GotoXY(59,KOUNT + 8);
          write(LENGTH_PACKET);
        end;
    end;
    if ESDBSREC.RecStatus=1 then
    begin
      writeln('InputRec RecStatus=1',
                      '  NumberRec=',RecordNumber);
    end;
    if (KOUNT=8) then begin
                KOUNT:=0;
                GotoXY(2,22);
                write('PRESS RETURN TO CONTINUE...');
                repeat
                 ch:=readkey;
                until (ch=#13);
                RestoreWindow(3,0,0);
                     end;
    end;
  end;
{ Initialisation routines for the variables defined     }
procedure INITVARNETSUM;{Routine #9}
 begin
      THRPUTCUR_MIN:=0;
      THRPUT_IN_PREV_MIN:=0;
      THRPUT_AVG_MN:=0;
      THRPUTPEK_OVER_HR:=0;
      UTILIS_IN_PREV_MIN:=0;
      UTILIS_AVG_MN:=0;
      UTILISPEK_OVER_HR:=0;
      RINGUPTIME:=0;
      RINGDOWNTIME:=0;
      SAMPLETIME:=1.0;
      PAKSUMCUR_MIN:=0;
      PAKSUM_IN_PREV_MIN:=0;
      PAKSUM_AVG_MN:=0;
      PAKSUMPEK_OVER_HR:=0;
      NO_OF_MINUTES:=0;
      MAX_BYTE_IN_MINUTE:=(MAX_BYTE_IN_SEC * 60);
 end;
 procedure INITVARASSEMBLY;{Routine #10}
 begin
      SECCOUNT:=0;
```

```
        MINCOUNT:=0;
        SECUPDATE:=0;
        MINUPDATE:=1;
        FIRST:=1;
        NEW_PKT_FLAG:=01;
        NO_OF_BYTES_SEC:=0;
        NO_OF_PACKETS_SEC:=0;
        INTM_BUF_PTR:=0;
        BYTEFLAG:=0;
  end;

procedure INITPROCESSECVAR;(Routine #11)
 begin
        PIPE_FLAG:=0;
        BUFF_PTR:=0;
        READ_BYTE:=0;
        UP_TP_PREV_SEC:=0;
        PIPE_PTR:=0;
        MATCH_INTM_BUF_PTR:=1;
        ALREADY_IN:=0;
        NO_KEY:=0;
        ASCII_VALUE:='0';
        KOUNT:=0;
 end;
 procedure INITDBASVAR;(Routine #12)
 begin
        THROUGHPUT:=0;
        PACKET:=0;
        RECORDS_IN_DBAS:=0;
        RECORDS_IN_NLDBAS:=0;
        RECORDS_IN_ESDBAS:=0;
 end;
 procedure INITNODELISTVAR;(Routine #13)
 var I:integer;
 begin
       for I:=0 to 1550 do NODE_LIST_ARRAY1[I]:=0;
       for I:=0 to 1550 do NODE_LIST_ARRAY2[I]:=0;
       NODE_LIST_PTR:=1;
       INDEX:=0;
       OFFSET_VALUE:=0;
       NODE_LIST_UPDATE_FLAG:=0;
 end;
 procedure INITERRORSTATVAR;(Routine #14)
 var I:integer;
 begin
       PAKINERROR:=0;
       CURRENT_ERROR_LIST_PTR:=0;
       NO_OF_TOKEN_LOSSES:=0;
 end;
```

```
procedure INITGRAPHICVAR;(Routine #15)
 var I:integer;
 begin
    for I:=1 to 60 do THRPUT_SEC[I,1]:=I;
    for I:=1 to 60 do THRPUT_SEC[I,2]:=0;
    THRPUT_SEC_PTR:=1;
    for I:=1 to 60 do UTILIS_SEC[I,1]:=I;
    for I:=1 to 60 do UTILIS_SEC[I,2]:=0;
    UTILIS_SEC_PTR:=1;
    for I:=1 to 60 do PAKSUM_SEC[I,1]:=I;
    for I:=1 to 60 do PAKSUM_SEC[I,2]:=0;
    PAKSUM_SEC_PTR:=1;
    for I:=1 to 60 do THRPUT_INTM_BUF[I]:=0;
    INTM_BUF_PTR:=0;
    for I:=1 to 60 do PAKSUM_INTM_BUF[I]:=0;
    for I:=1 to 60 do THRPUT_OVER_HR[I]:=0;
    THRPUT_MIN_PTR:=1;
    for I:=1 to 60 do UTILIS_OVER_HR[I]:=0;
    UTILIS_MIN_PTR:=1;
    for I:=1 to 60 do PAKSUM_OVER_HR[I]:=0;
    PAKSUM_MIN_PTR:=1;
 end;
 { procedure which displays  those parameters of
   NETWORK SUMMARY, that need to be every minute }
procedure WRITEDISPMIN1;(Routine #16)
     begin
          GotoXY(14,8);
          write(THRPUT_IN_PREV_MIN);
          GotoXY(26,8);
          write(THRPUT_AVG_MN);
          GotoXY(37,8);
          write( THRPUTPEK_OVER_HR);
          GotoXY(45,8);
          write( UTILIS_IN_PREV_MIN:7:5);
          GotoXY(56,8);
          write( UTILIS_AVG_MN:7:5);
          GotoXY(67,8);
          write( UTILISPEK_OVER_HR:7:5);
          GotoXY(30,14);
          write( SAMPLETIME:4:2);
          GotoXY(30,16);
          write(RECORDS_IN_DBAS);
          GotoXY(30,18);
          write(RECORDS_IN_NLDBAS);
          GotoXY(30,20);
          write(RECORDS_IN_ESDBAS);
          GotoXY(74,17);
          write( PAKSUM_IN_PREV_MIN);
          GotoXY(74,19);
          write( PAKSUM_AVG_MN);                    end;
```

```
    { procedure which displays those parameters of
      NETWORK SUMMARY, that need to be every second.   }
procedure WRITEDISPSEC1;{Routine #17}
     begin
          GotoXY(4,8);
          write( THRPUTCUR_MIN);
          GotoXY(74,15);
          write( PAKSUMCUR_MIN);
     end;
   { procedure which is invoked by the procedure that
     displays on demand the parameters of
     NODE LIST STATISTICS  }
procedure DISPLAY_NODE_LIST;{Routine #18}
     begin
         if I<20 then
         begin
              GotoXY(3,I); write(NLDBSREC.NODEAD);
              GotoXY(16,I);write(NLDBSREC.NUMBERBYTE);
              GotoXY(33,I); write(NLDBSREC.NUMBERPAK);
              GotoXY(47,I);write(NLDBSREC.AVERAGESIZE);
              GotoXY(59,I); write(NLDBSREC.PAKRCV);
              GotoXY(69,I);write(NLDBSREC.RECEIVEDNO);
              inc (I);
          end;
     end;
```

{procedures which process NODE_LIST_ARRAYs 1 & 2 respectively,indexing the array through node address . If the first byte i.e. DIRTY byte is 1 then it was involved in transmission to some other node in the last one minute,if 3 then was involved only in receiving of data from some node, else it neither transmitted any data nor did it receive any data. }

```
procedure PROCESS_NODE_LIST1;{Routine #19}
     begin
       OFFSET_VALUE:=1+(INDEX-1)*5;
       if (NODE_LIST_ARRAY1[OFFSET_VALUE]=1)
       then begin
        NLDBSREC.NODEAD:=INDEX;
        Inc (OFFSET_VALUE);
        NLDBSREC.NUMBERPAK:=NODE_LIST_ARRAY1[OFFSET_VALUE];
        Inc (OFFSET_VALUE);
        NLDBSREC.PAKRCV:=NODE_LIST_ARRAY1[OFFSET_VALUE];
        Inc (OFFSET_VALUE);
        NLDBSREC.NUMBERBYTE:=NODE_LIST_ARRAY1[OFFSET_VALUE];
        Inc (OFFSET_VALUE);
        NLDBSREC.AVERAGESIZE:=
              Round(NLDBSREC.NUMBERBYTE/NLDBSREC.NUMBERPAK);
        NLDBSREC.RECEIVEDNO:=NODE_LIST_ARRAY1[OFFSET_VALUE];
```

```
        if (NODE_LIST_UPDATE_FLAG=1) then
             begin
                 NODELISTDBSUPDATE;
             end
        else DISPLAY_NODE_LIST;
        end;
        if (NODE_LIST_ARRAY1[OFFSET_VALUE]=3)  then
        begin
         NLDBSREC.NODEAD:=INDEX;
         Inc (OFFSET_VALUE);
         NLDBSREC.NUMBERPAK:=NODE_LIST_ARRAY1[OFFSET_VALUE];
        Inc (OFFSET_VALUE);
        NLDBSREC.PAKRCV:=NODE_LIST_ARRAY1[OFFSET_VALUE];
        Inc (OFFSET_VALUE);
        NLDBSREC.NUMBERBYTE:=NODE_LIST_ARRAY1[OFFSET_VALUE];
        Inc (OFFSET_VALUE);
        NLDBSREC.AVERAGESIZE:=0;
        NLDBSREC.RECEIVEDNO:=NODE_LIST_ARRAY1[OFFSET_VALUE];
        if (NODE_LIST_UPDATE_FLAG=1) then
        begin
           NODELISTDBSUPDATE;
        end
        else DISPLAY_NODE_LIST;
        end;
     end;

procedure PROCESS_NODE_LIST2;{Routine #20}
  begin
   OFFSET_VALUE:=1+(INDEX-1)*5;
   if (NODE_LIST_ARRAY2[OFFSET_VALUE]=1)
   then begin
       NLDBSREC.NODEAD:=INDEX;
       Inc (OFFSET_VALUE);
       NLDBSREC.NUMBERPAK:=NODE_LIST_ARRAY2[OFFSET_VALUE];
       Inc (OFFSET_VALUE);
       NLDBSREC.PAKRCV:=NODE_LIST_ARRAY2[OFFSET_VALUE];
       Inc (OFFSET_VALUE);
       NLDBSREC.NUMBERBYTE:=NODE_LIST_ARRAY2[OFFSET_VALUE];
       Inc (OFFSET_VALUE);
       NLDBSREC.AVERAGESIZE:=
       Round(NLDBSREC.NUMBERBYTE/NLDBSREC.NUMBERPAK);
       NLDBSREC.RECEIVEDNO:=NODE_LIST_ARRAY2[OFFSET_VALUE];
       if (NODE_LIST_UPDATE_FLAG=1) then
       begin
          NODELISTDBSUPDATE;
       end
       else DISPLAY_NODE_LIST;
       end;
       if (NODE_LIST_ARRAY2[OFFSET_VALUE]=3)  then
       begin
```

```
        NLDBSREC.NODEAD:=INDEX;
        Inc (OFFSET_VALUE);
        NLDBSREC.NUMBERPAK:=NODE_LIST_ARRAY2[OFFSET_VALUE];
        Inc (OFFSET_VALUE);
        NLDBSREC.PAKRCV:=NODE_LIST_ARRAY2[OFFSET_VALUE];
        Inc (OFFSET_VALUE);
        NLDBSREC.NUMBERBYTE:=NODE_LIST_ARRAY2[OFFSET_VALUE];
        Inc (OFFSET_VALUE);
        NLDBSREC.AVERAGESIZE:=0;
        NLDBSREC.RECEIVEDNO:=NODE_LIST_ARRAY2[OFFSET_VALUE];
        if (NODE_LIST_UPDATE_FLAG=1) then
        begin
            NODELISTDBSUPDATE;
        end
        else DISPLAY_NODE_LIST;
       end;
end;

{ procedure which displays the parameters of
          NODE LIST STATISTICS on demand}
procedure WRITEDISP2;{Routine #21}
     begin
       I:=7;
       for  INDEX:=0 to 255 do
       begin
         if NODE_LIST_PTR=2 then begin
                                   PROCESS_NODE_LIST1;
                                  end
       else   begin
                PROCESS_NODE_LIST2;
              end;
       end;
     end;

procedure WRITEDISPSEC3;{Routine #22}
 begin
   GotoXY(36,4);
   write(PAKINERROR);
 end;


{ Interrupt Service routine of INT1Ch that gets executed
                                     every 55.54msec.}
procedure Int1CHandler(Flags,CS,IP,AX,BX,CX,DX,SI,
                                    DI,DS,ES,BP:word);
{Routine #23}
interrupt;
begin
   Inc(SECCOUNT);
   if SECCOUNT=18 then
```

```
      DBSREC.RecStatus:=00;
      DBSREC.DBSDATE:=Concat(STMM,':',STDD,':',STYYYY);
      NumberRec:=0;
      AddRec(dbas,NumberRec,DBSREC);
      RECORDS_IN_DBAS:=NumberRec;
      THRPUTCUR_MIN:=0;
      PAKSUMCUR_MIN:=0;
      UTILIS_IN_PREV_MIN:=(THRPUT_IN_PREV_MIN)/
                               (MAX_BYTE_IN_MINUTE);
      if (THRPUT_IN_PREV_MIN > THRPUTPEK_OVER_HR) then
      THRPUTPEK_OVER_HR:=THRPUT_IN_PREV_MIN;
      if (UTILIS_IN_PREV_MIN > UTILISPEK_OVER_HR) then
          UTILISPEK_OVER_HR:=UTILIS_IN_PREV_MIN;
      MATCH_INTM_BUF_PTR:=1;
   end;
 end;
end;
{ procedure that gets executed whenever MINUPDATE is found
                                            to be set to 1 }
procedure PROCESS_EVERY_MINUTE;(Routine #25)
begin
     Inc (NO_OF_MINUTES);
     THRPUT_AVG_MN:=((THRPUT_IN_PREV_MIN) div 60);
     UTILIS_AVG_MN:=(THRPUT_IN_PREV_MIN)/
                                (MAX_BYTE_IN_MINUTE);
     PAKSUM_AVG_MN:=((PAKSUM_IN_PREV_MIN) div 60);
end;
{ procedure that transfers the data required to draw the
Graph of a parameter in last 60 seconds.  }
procedure TRANSFERVALUES_THRPUT;(Routine #26)
var TEMP_INTM_BUF_PTR:integer;
begin
     I:=60;
     TEMP_INTM_BUF_PTR:=INTM_BUF_PTR;
     while (I <> 0)  do
     begin
       THRPUT_SEC[I,2]:=THRPUT_INTM_BUF[TEMP_INTM_BUF_PTR];
       I:=I - 1;
       TEMP_INTM_BUF_PTR:=TEMP_INTM_BUF_PTR - 1;
       if TEMP_INTM_BUF_PTR = 0 then
       TEMP_INTM_BUF_PTR:=60;
     end;
 end;
 procedure TRANSFERVALUES_PACKET;(Routine #27)
 var TEMP_INTM_BUF_PTR:integer;
begin
     I:=60;
     TEMP_INTM_BUF_PTR:=INTM_BUF_PTR;
     while (I <> 0)  do
     begin
```

```
      PAKSUM_SEC[I,2]:=PAKSUM_INTM_BUF[TEMP_INTM_BUF_PTR];
      I:=I - 1;
      TEMP_INTM_BUF_PTR:=TEMP_INTM_BUF_PTR - 1;
      if TEMP_INTM_BUF_PTR = 0 then
      TEMP_INTM_BUF_PTR:=60;
    end;
 end;

{ procedure that displays the VARIABLE DUMP    }
procedure UPDATE_DEBUG_VALUES;{Routine #28}
begin

      GotoXY(32,3);
      write(NO_OF_PACKETS_SEC);
      GotoXY(32,4);
      write(NO_OF_BYTES_SEC);
      GotoXY(32,6);
      write(INTM_BUF_PTR);
      GotoXY(34,7);
      write(THRPUT_INTM_BUF[INTM_BUF_PTR]);
      GotoXY(34,8);
      write(PAKSUM_INTM_BUF[INTM_BUF_PTR]);
      GotoXY(32,12);
      write(THRPUTCUR_MIN);
      GotoXY(32,13);
      write(THRPUT_IN_PREV_MIN);
      GotoXY(32,14);
      write(THRPUT_AVG_MN);
      GotoXY(32,15);
      write(THRPUTPEK_OVER_HR);
      GotoXY(32,17);
      write(UTILIS_IN_PREV_MIN);
      GotoXY(32,18);
      write(UTILIS_AVG_MN);
      GotoXY(32,19);
      write(UTILISPEK_OVER_HR);
      GotoXY(32,21);
      write(PAKSUMCUR_MIN);
      GotoXY(32,22);
      write(PAKSUM_IN_PREV_MIN);
      GotoXY(32,23);
      write(PAKSUM_AVG_MN);
      GotoXY(32,24);
      write(PAKSUMPEK_OVER_HR);
      GotoXY(60,3);
      write(MATCH_INTM_BUF_PTR);
      GotoXY(62,10);
      write(NODE_LIST_PTR);
      GotoXY(62,7);
      write(THRPUT_INTM_BUF[MATCH_INTM_BUF_PTR]);
```

```
        GotoXY(62,8);
        write(PAKSUM_INTM_BUF[MATCH_INTM_BUF_PTR]);
        GotoXY(62,13);
        write(BUFF_PTR,'  ');
        GotoXY(62,14);
        write(PIPE_PTR,'  ');
end;
{ procedure which loads the proper HELP window on demand.  }
procedure HELP_PROCESS;{Routine #29}
begin
        case PREV_FUNCT_KEY of
        2:RestoreWindow(9,0,0);
        3:RestoreWindow(10,0,0);
        4:RestoreWindow(11,0,0);
        5:RestoreWindow(8,0,0);
        6:begin end;
        7:begin end;
        8:begin end;
        9:begin end;
        10:begin end;
        end;
end;
procedure FILE_IN_PROMPT;{Routine #30}
begin
  GotoXY(2,22);
  write('                                                    ');
  GotoXY(2,22);
  write('Enter valid Filename(in the form filename.ext):
  COLNO:=50;
  ROWNO:=22;
  PRINT_I:=1;
  PTR:=1;
  ALREADY_IN:=1;
end;
procedure MESSAGE_PROMPT;{Routine #31}
begin
  GotoXY(2,22);
  write('Do you want to SAVE in a file or PRINT?
                    (Press S for SAVE,P for PRINT)');
end;

procedure READ_FILE_NAME;{Routine #32}
begin
  repeat
  ASCII_VALUE:=readkey;
  if ((ASCII_VALUE>#45) and (ASCII_VALUE<#58))
        or ((ASCII_VALUE>#96) and (ASCII_VALUE<#123))
        or ((ASCII_VALUE>#64) and (ASCII_VALUE<#91)) THEN
       begin
         FNAME[PTR]:=ASCII_VALUE;
```

```
            Inc (PTR);
            GotoXY(COLNO,ROWNO);
            write(ASCII_VALUE);
            inc (COLNO);
            PRINT_I:=PRINT_I + 1;
          end;
          until ((ASCII_VALUE=#13) or (PRINT_I>12));
          GotoXY(2,22);
          write('                                           ');
          FillChar(fname_str,SizeOf(fname_str),' ');
          fname[0]:='0';
          fname_str:=Copy(fname,1,(PTR-1));
end;
procedure PRINTFACILITY;(Routine #33)
begin
  MESSAGE_PROMPT;
  repeat
  ASCII_VALUE:=readkey;
  until (ASCII_VALUE='S') or (ASCII_VALUE = 's') or
        (ASCII_VALUE ='p') or (ASCII_VALUE = 'P');
  begin
    if ((ASCII_VALUE='p') or (ASCII_VALUE='P'))  then
    begin
      GotoXY(2,22);
      write('                                           ');
      begin
        GotoXY(2,22);
        write('                                         ');
        repeat
        until (keypressed=true);
        HardCopy(false,6);
        ALREADY_IN:=0;
      end;
    end;
    if ((ASCII_VALUE='s') or (ASCII_VALUE='S')) then
                                          FILE_IN_PROMPT;
  end;
  if ((ASCII_VALUE='s') or (ASCII_VALUE='S')) then
  begin
    READ_FILE_NAME;
    SaveScreen(fname_str);
    if GetErrorCode <> -1 then  FILE_IN_PROMPT;
  end;
  end;
procedure show;(Routine #34)
var
   DirInfo:SearchRec;
begin
  ClearScreen;
  writeln;
```

```
  GotoXY(3,21);
  write(' Key in the File name of the file in which a
                                          screen is stored');
  FILE_IN_PROMPT;
  READ_FILE_NAME;
  FindFirst(fname_str,Archive,DirInfo);
  if DosError=0 then
  begin
    LoadScreen(fname_str);
    GotoXY(3,22);
    write('PRESS RETURN TO CONTINUE.....');
    repeat
      ch:=readkey;
    until (ch=#13);
    RestoreWindow(1,0,0);
    FUNCT_KEY:=2;
  end;
  if DosError=2 then begin
          GotoXY(2,22);
          write('FILE NOT FOUND IN CURRENT DIRECTORY   ');
          repeat
            ch:=readkey;
          until(ch=#13);
                    end;
  if DosError=18 then begin
          GotoXY(2,22);
          write('NO MORE FILES                          ');
          repeat
            ch:=readkey;
          until(ch=#13);
                       end;
end;
procedure DISPGRAPHIC;(Routine #35)
var dx,dy,x1,y1,x2,y2,Lines,Scale:integer;
begin
    ClearScreen;
    DefineWindow(4,0,0,XMaxGlb,YMaxGlb);
    DefineWorld(4,0,0,719,349);
    SelectWorld(4);
    SelectWindow(4);
    DrawBorder;
    DrawText(260,10,2,'GRAPHIC DISPLAY ');
    SetLineStyle(0);
    DrawStraight(0,719,285);
    DrawStraight(0,719,28);
    MENU2;
end;
procedure THRPUT_GRAPH;(Routine #36)
begin
    DefineWindow(6,0,30,XMaxGlb,286);
```

```
    DefineHeader(6,'THROUGHPUT CURVE');
    SelectWindow(6);
    SetHeaderOn;
    DefineWorld(6,0,0,60,1800);
    SelectWorld(6);
    DrawAxis(9,9,5,12,5,18,0,0,true);
    DrawPolygon(THRPUT_SEC,1,60,7,2,0);
    GotoXY(2,8);
    write('T');
    GotoXY(2,9);
    write('H');
    GotoXY(2,10);
    write('R');
    GotoXY(2,11);
    write('O');
    GotoXY(2,12);
    write('U');
    GotoXY(2,13);
    write('G');
    GotoXY(2,14);
    write('H');
    GotoXY(2,15);
    write('P');
    GotoXY(2,16);
    write('U');
    GotoXY(2,17);
    write('T');
    GotoXY(4,9);
    write('B');
    GotoXY(4,10);
    write('Y');
    GotoXY(4,11);
    write('T');
    GotoXY(4,12);
    write('E');
    GotoXY(4,13);
    write('S');
    GotoXY(4,14);
    write('/');
    GotoXY(4,15);
    write('S');
    GotoXY(4,16);
    write('E');
    GotoXY(4,17);
    write('C');
    GotoXY(20,20);
    write('<----------- T I M E(in seconds)------------->');
    DrawBorder;
end;
```

```
procedure PACKET_GRAPH;(Routine #37)
begin
    DefineWindow(7,0,30,XMaxGlb,286);
    DefineHeader(7,'PACKET  CURVE');
    SelectWindow(7);
    SetHeaderOn;
    DefineWorld(4,0,0,60,1800);
    SelectWorld(4);
    DrawAxis(9,9,5,12,5,18,0,0,true);
    DrawPolygon(THRPUT_SEC,1,60,7,2,0);
    GotoXY(4,8);
    write('P');
    GotoXY(4,9);
    write('A');
    GotoXY(4,10);
    write('C');
    GotoXY(4,11);
    write('K');
    GotoXY(4,12);
    write('E');
    GotoXY(4,13);
    write('T');
    GotoXY(4,14);
    write('S');
    GotoXY(4,15);
    write('/');
    GotoXY(4,16);
    write('S');
    GotoXY(4,17);
    write('E');
    GotoXY(4,18);
    write('C');
    GotoXY(20,20);
    write('<------------ T I M E(in seconds)------------->');
    DrawBorder;
end;

procedure RAMSCREENSETUP;(Routine #38)
var ch:char;ArcX1,ArcY1,ArcX2,ArcY2:Float;
begin
    InitGraphic;
    DrawBorder;
    DrawStraight(109,629,50);
    DrawLine(109,50,109,174);
    DrawStraight(109,629,174);
    DrawLine(629,174,629,50);
    DrawLine(405,174,405,209);
    DrawLine(355,209,355,244);
    DrawLine(355,244,455,244);
    DrawLine(455,244,455,209);
```

```
DrawLine(455,209,355,209);
DrawLine(405,244,405,269);
DrawLine(405,269,249,269);
DrawLine(249,290,249,259);
DrawLine(239,199,249,225);
DrawLine(249,225,249,254);
DrawLine(224,209,109,209);
DrawLine(109,209,129,199);
DrawLine(129,199,239,199);
DrawLine(239,199,224,209);
DrawLine(109,209,109,274);
DrawLine(59,310,224,310);
DrawLine(224,310,224,279);
DrawLine(224,310,249,290);
DrawLine(59,279,224,279);
DrawLine(59,279,59,310);
DrawLine(59,279,89,259);
DrawLine(89,259,109,259);
DrawLine(249,259,245,259);
DrawLine(224,274,224,209);
DrawLine(109,274,224,274);
DrawLine(224,274,249,254);
DrawLine(224,279,249,259);
DrawLine(178,285,218,285);
DrawLine(218,285,218,305);
DrawLine(218,295,178,295);
DrawLine(178,305,218,305);
DrawLine(178,305,178,285);
DrawLine(181,300,215,300);
DrawLine(109,209,118,216);
DrawLine(224,209,215,216);
DrawLine(109,274,118,268);
DrawLine(224,274,215,268);
DrawLine(39,339,59,314);
DrawLine(59,314,229,314);
DrawLine(229,314,204,339);
DrawLine(204,339,39,339);
DrawLine(39,342,204,342);
DrawLine(39,339,39,342);
DrawLine(204,339,204,342);
DrawLine(204,342,229,320);
DrawLine(229,320,229,314);
DrawText(59,324,1,' PRESS <CR> TO CONTINUE');
DrawText(365,218,1,'IITK LAN CARD');
DrawSquare(118,216,215,268,true);
DrawText(189,74,2,'   TOKEN RING NETWORK');
DrawText(260,260,1,'HEADER(FOUR BYTES) OF A');
DrawText(260,273,1,'PACKET SENT TO MONITOR');
SetColorBlack;
DrawText(104,224,1,'    PC - BASED');
```

```
DrawText(104,259,1,'   NETWORK MONITOR');
SetColorWhite;
repeat
   ch:=Readkey;
until (ch=#13);
ResetWindowStack;
ClearScreen;
OVER_ALL_HELP;
StoreWindow(8);
GotoXY(3,23);
write('PRESS RETURN TO CONTINUE.....');
repeat
   ch:=readkey;
until (ch=#13);

ClearScreen;
HELPSCREEN1;
StoreWindow(9);
GotoXY(3,22);
write('PRESS RETURN TO CONTINUE.....');
repeat
    ch:=readkey;
until (ch=#13);

ClearScreen;
DISPNETWORKSUMMARY;
StoreWindow(1);
GotoXY(3,22);
write('PRESS RETURN TO CONTINUE.....');
repeat
    ch:=readkey;
until (ch=#13);

ClearScreen;
HELPSCREEN2;
StoreWindow(10);
GotoXY(3,22);
write('PRESS RETURN TO CONTINUE.....');
repeat
   ch:=readkey;
until (ch=#13);
ClearScreen;
DISPNODELISTSTAT;
StoreWindow(2);
GotoXY(3,22);
write('PRESS RETURN TO CONTINUE.....');
repeat
   ch:=readkey;
until (ch=#13);
```

```
     ClearScreen;
     HELPSCREEN3;
     StoreWindow(11);
     GotoXY(3,22);
     write('PRESS RETURN TO CONTINUE.....');
     repeat
        ch:=readkey;
     until (ch=#13);

     ClearScreen;
     DISPERRORSTAT;
     StoreWindow(3);
     GotoXY(3,22);
     write('PRESS RETURN TO CONTINUE.....');
     repeat
        ch:=readkey;
     until (ch=#13);
     ClearScreen;
     DISPGRAPHIC;
     StoreWindow(4);
     THRPUT_GRAPH;
     StoreWindow(6);
     GotoXY(3,22);
     write('PRESS RETURN TO CONTINUE.....');
     repeat
        ch:=readkey;
     until (ch=#13);
     ClearScreen;
     PACKET_GRAPH;
     StoreWindow(7);
     GotoXY(3,22);
     write('PRESS RETURN TO CONTINUE.......');
     repeat
        ch:=readkey;
     until (ch=#13);
     LeaveGraphic;

end;
procedure DISPLAY_NODE_LIST1;{Routine #39}
var i,j:integer;
begin
     j:=4;
     GotoXY(8,j);
     write('ND DB SR DST LEN of NODE_LIST_ARRAY1');
     for i:=5 to 15 do
     begin
          GotoXY(8,i);
          write(i-4);
     end;
     j:=5;
```

```
    for i:=1 to 10 do
    begin
    GotoXY(12,j);
    write(NODE_LIST_ARRAY1[1+(i-1)*4],'  ',
    NODE_LIST_ARRAY1[2+(i-1)*4],'  ',NODE_LIST_ARRAY1[3+(i-1)*4],
    '  ',NODE_LIST_ARRAY1[4+(i-1)*4]);
    Inc (j);
 end;
end;
procedure DISPLAY_NODE_LIST2;(Routine #40)
var i,j:integer;
begin
    j:=4;
    GotoXY(8,j);
    write('ND DB SR DST LEN of NODE_LIST_ARRAY2');
    for i:=5 to 15 do
    begin
         GotoXY(8,i);
         write(i-4);
    end;
    j:=5;
    for i:=1 to 10 do
    begin
    GotoXY(12,j);
    write(NODE_LIST_ARRAY2[1+(i-1)*4],'  ',
    NODE_LIST_ARRAY2[2+(i-1)*4],'  ',NODE_LIST_ARRAY2[3+(i-1)*4],
    '  ',NODE_LIST_ARRAY2[4+(i-1)*4]);
    Inc (j);
    end;
end;
procedure INITIALISE_VARIABLES(Routine #41)
begin
 INITVARNETSUM;
 INITVARASSEMBLY;
 INITPROCESSECVAR;
 INITDBASVAR;
 INITNODELISTVAR;
 INITERRORSTATVAR;
 INITGRAPHICVAR;
 ClrScr;
end;

procedure SETUP_RAMSCREENS(Routine #42)
begin
 RAMSCREENSETUP;
 GetIntVec($1C,Int1CSave);
 SetIntVec($1C,@Int1CHandler);
 NetSum_DataBase_Build;
 NodeList_DataBase_Build;
 ErrorStat_DataBase_Build;
```

```
 INITIRQ2;
 EnterGraphic;
 RestoreWindow(8,0,0);
end;
procedure UPDATE_DATE;(Routine #43}
begin
  GetDate(YYYY,MM,DD,DOFW);
  GotoXY(65,2);
  write(MM,':',DD,':',YYYY);
end;
procedure ON_SECUPDATE;(Routine #44}
 begin
      SECUPDATE:=0;
      PROCESS_INTM_BUFF_SEC;
      GotoXY(6,2);
      write('        ');
      SC:=00;
      GetTime(HR,MN,SC,SCHUN);
      GotoXY(6,2);
      write(HR,':',MN,':',SC);
      case FUNCT_KEY of
      1:begin end;
      2:WRITEDISPSEC1;
      4:WRITEDISPSEC3;
      5:begin
         if SKIP_FLAG<>1 then begin
              TRANSFERVALUES_THRPUT; ResetAxis;
              DrawPolygon(THRPUT_SEC,1,60,7,2,0);
              SKIP_FLAG:=1;  end;
        end;
      6:begin end;
      7:begin end;
      8:begin
         if SKIP_FLAG<>1 then begin
              TRANSFERVALUES_PACKET; ResetAxis;
              DrawPolygon(PAKSUM_SEC,1,60,7,2,0);
              SKIP_FLAG:=1;   end;
        end;
     10:begin
         UPDATE_DEBUG_VALUES;
        end;
   end;
end;
procedure ON_MINUPDATE; (Routine #45}
begin
 if NODE_LIST_PTR=1 then begin
                 NODE_LIST_PTR:=2;
                 for INDEX:=0 to 1550 do
                 NODE_LIST_ARRAY2[INDEX]:=0;
                            end
```

```
     else begin
                    NODE_LIST_PTR:=1;
                    for INDEX:=0 to 1550 do
                    NODE_LIST_ARRAY1[INDEX]:=0;
          end;
    MINUPDATE:=0;
    PROCESS_EVERY_MINUTE;
    NODE_LIST_UPDATE_FLAG:=1;
    for INDEX:=0 to 255 do
        begin
          if NODE_LIST_PTR=2 then PROCESS_NODE_LIST1
                                else PROCESS_NODE_LIST2;
        end;
    NODE_LIST_UPDATE_FLAG:=0;
    case FUNCT_KEY of
          1:begin end;
          2:begin
               RestoreWindow(1,0,0);
               WRITEDISPMIN1;
            end;
          3:begin
               RestoreWindow(2,0,0);
               WRITEDISP2;
            end;
          4:begin
              RestoreWindow(3,0,0);
            end;
          5:begin end;
          6:begin end;
          7:begin end;
          8:begin end;
          9:begin end;
         10:begin
              CLEANSCREEN_VALUES;
              UPDATE_DEBUG_VALUES;
            end;
     end;
end;
procedure UPDATE_ESDBAS;{Routine #46}
begin
  ESDBSREC.RecStatus:=00;
      while (BUFF_PTR<>PIPE_PTR) and (PIPE_FLAG<>0) do
      begin
      READ_BYTE:=RINGBUFF[PIPE_PTR];
      if PIPE_FLAG=1 then
                       ESDBSREC.SOURCE_ADDRESS:=READ_BYTE
      if PIPE_FLAG=2 then ESDBSREC.DEST_ADDRESS:=READ_BYTE;
      if PIPE_FLAG=3 then
         begin
            ESDBSREC.LENGTH_PACKET:=READ_BYTE;
```

```
            ERRORSTATDBSUPDATE;
            ERROR_FLAG:=0;
          end;
        Inc (PIPE_PTR);
        if (PIPE_PTR=32000) then PIPE_PTR:=0;
        Inc (PIPE_FLAG);
        if PIPE_FLAG=4 then PIPE_FLAG:=0;
        end;
end;
procedure UPDATE_NODELIST;{Routine #47}
begin
 while (BUFF_PTR<>PIPE_PTR) and (PIPE_FLAG<>0) do
        begin
          READ_BYTE:=RINGBUFF[PIPE_PTR];
          if (PIPE_FLAG=1) then SOURCEAD:=READ_BYTE;
          if (PIPE_FLAG=2) then DESTAD:=READ_BYTE;
          if (PIPE_FLAG=3) then
               begin
                    LENGTH:=READ_BYTE;
                    HEADER_STATUS:=1;
               end;
          Inc(PIPE_PTR);
          if (PIPE_PTR=32000) then PIPE_PTR:=0;
          Inc(PIPE_FLAG);
          if (PIPE_FLAG=4) then PIPE_FLAG:=0;
        end;
        if HEADER_STATUS=1 then
        begin
          if NODE_LIST_PTR=1 then
          begin
            HEADER_STATUS:=0;
            NODE_LIST_ARRAY1[1+(SOURCEAD-1)*5]:=1;
            NODE_LIST_ARRAY1[2+(SOURCEAD-1)*5]:=
                 NODE_LIST_ARRAY1[2+(SOURCEAD-1)*5] +1;
            NODE_LIST_ARRAY1[4+(SOURCEAD-1)*5]:=
                 NODE_LIST_ARRAY1[4+(SOURCEAD-1)*5]+LENGTH;
           if NODE_LIST_ARRAY1[1+(DESTAD-1)*5]=0 then
           begin
             NODE_LIST_ARRAY1[1+(DESTAD-1)*5]:=3;
             NODE_LIST_ARRAY1[2+(DESTAD-1)*5]:=0;
             NODE_LIST_ARRAY1[4+(DESTAD-1)*5]:=0;
             NODE_LIST_ARRAY1[3+(DESTAD-1)*5]:=1;
             NODE_LIST_ARRAY1[5+(DESTAD-1)*5]:=LENGTH;
           end else
           if NODE_LIST_ARRAY1[1+(DESTAD-1)*5]=3 then
           begin  NODE_LIST_ARRAY1[3+(DESTAD-1)*5]:=
                    NODE_LIST_ARRAY1[3+(DESTAD-1)*5]+1;
                  NODE_LIST_ARRAY1[5+(DESTAD-1)*5]:=
                  NODE_LIST_ARRAY1[5+(DESTAD-1)*5]+LENGTH;
           end
```

```
        else begin
              NODE_LIST_ARRAY1[3+(DESTAD-1)*5]:=
                 NODE_LIST_ARRAY1[3+(DESTAD-1)*5] +1;
              NODE_LIST_ARRAY1[5+(DESTAD-1)*5]:=
              NODE_LIST_ARRAY1[5+(DESTAD-1)*5]+LENGTH;
             end;
        end;
        if NODE_LIST_PTR=2 then
        begin
            HEADER_STATUS:=0;
            NODE_LIST_ARRAY2[1+(SOURCEAD-1)*5]:=1;
            NODE_LIST_ARRAY2[2+(SOURCEAD-1)*5]:=
                NODE_LIST_ARRAY2[2+(SOURCEAD-1)*5] +1;
            NODE_LIST_ARRAY2[4+(SOURCEAD-1)*5]:=
            NODE_LIST_ARRAY2[4+(SOURCEAD-1)*5] + LENGTH;
            if NODE_LIST_ARRAY2[1+(DESTAD-1)*5]=0 then
            begin
                NODE_LIST_ARRAY2[1+(DESTAD-1)*5]:=3;
                NODE_LIST_ARRAY2[2+(DESTAD-1)*5]:=0;
                NODE_LIST_ARRAY2[4+(DESTAD-1)*5]:=0;
                NODE_LIST_ARRAY2[3+(DESTAD-1)*5]:=1;
                NODE_LIST_ARRAY2[5+(DESTAD-1)*5]:=LENGTH;
            end else
            if NODE_LIST_ARRAY2[1+(DESTAD-1)*5]=3 then
            begin
               NODE_LIST_ARRAY2[3+(DESTAD-1)*5]:=
                NODE_LIST_ARRAY2[3+(DESTAD-1)*5] + 1;
               NODE_LIST_ARRAY2[5+(DESTAD-1)*5]:=
                NODE_LIST_ARRAY2[5+(DESTAD-1)*5] + LENGTH;
            end
            else begin
                NODE_LIST_ARRAY2[3+(DESTAD-1)*5]:=
                 NODE_LIST_ARRAY1[3+(DESTAD-1)*5] +1;
                NODE_LIST_ARRAY2[5+(DESTAD-1)*5]:=
                 NODE_LIST_ARRAY2[5+(DESTAD-1)*5] + LENGTH;
                 end;
            end;
         end;
      end;

end;
begin { M A I N }
INITIALISE_VARIABLES;
INITDATETIME;
SETUP_RAMSCREENS;
/--------------------------------------------------------------\
|if KeyPressed then                         {Routine 48}        |
|begin                                                          |
|  200:read(Kbd,ch);                                            |
|     if (ch=#27) and KeyPressed then
```

```
begin
   read(Kbd,ch);
     case ch of
       #59:begin PREV_FUNCT_KEY:=FUNCT_KEY;
                 FUNCT_KEY:=1; end;
       #60:begin PREV_FUNCT_KEY:=FUNCT_KEY;
                 FUNCT_KEY:=2; end;
       #61:begin PREV_FUNCT_KEY:=FUNCT_KEY;
                 FUNCT_KEY:=3; end;
       #62:begin PREV_FUNCT_KEY:=FUNCT_KEY;
                 FUNCT_KEY:=4; end;
       #63:begin PREV_FUNCT_KEY:=FUNCT_KEY;
                 FUNCT_KEY:=5; end;
       #64:begin PREV_FUNCT_KEY:=FUNCT_KEY;
                 FUNCT_KEY:=6; end;
       #65:begin PREV_FUNCT_KEY:=FUNCT_KEY;
                 FUNCT_KEY:=7; end;
       #66:begin PREV_FUNCT_KEY:=FUNCT_KEY;
                 FUNCT_KEY:=8; end;
       #67:begin PREV_FUNCT_KEY:=FUNCT_KEY;
                 FUNCT_KEY:=9; end;
       #68:begin PREV_FUNCT_KEY:=FUNCT_KEY;
                 FUNCT_KEY:=10; end;
       #84:goto 500;
       #89:show;
       #92:DISPNLDBSFILE;
       #112:begin ClearScreen; DISPESDBSFILE; end;
     end;
```

```
     case FUNCT_KEY of
       1:HELP_PROCESS;                       {Routine 49}
       2:begin RestoreWindow(1,0,0);
               WRITEDISPSEC1; WRITEDISPMIN1; end;
       3:begin RestoreWindow(2,0,0);WRITEDISP2;end;
       4:begin RestoreWindow(3,0,0);
               DISPESDBSFILE; end;
       5:begin RestoreWindow(4,0,0);
               RestoreWindow(6,0,0); ResetAxis;
               SKIP_FLAG:=0; end;
       6:PRINTFACILITY;
       7:begin DOSCOMMAND;
            if ((PREV_FUNCT_KEY < 5) and
                (PREV_FUNCT_KEY > 1)) then
               begin
                   FUNCT_KEY:=PREV_FUNCT_KEY;
                   RestoreWindow(FUNCT_KEY-1,0,0);
               end
            else begin
                   RestoreWindow(1,0,0);
                   FUNCT_KEY:=1;
```

```
                                    end
                              end;
                      8:begin RestoreWindow(7,0,0);
                                    ResetAxis;
                                    SKIP_FLAG:=0;
                        end;
                      9:DISPDBSFILE;
                     10:DEBUGROUTINEPROMPT;
                    end;
          end;
end;
UPDATE_DATE;

while (keypressed=false) do
begin
  if SECUPDATE=1 then
  begin
       ON_SECUPDATE;
  end;
  if MINUPDATE=1 then
  begin
     GotoXY(65,2);
     write(MM,':',DD,':',YYYY);
     ON_MINUPDATE;
  end;
  KOUNT:=0;
  while((BUFF_PTR>PIPE_PTR + 3) or (PIPE_PTR-BUFF_PTR>3))
   do begin
      READ_BYTE:=RINGBUFF[PIPE_PTR];
      if ((PIPE_FLAG=0) and (READ_BYTE<>02)) then
         begin
              inc (KOUNT);
              GotoXY(55,22);                  {Routine 50}
              write(KOUNT);
         end;
      if ((READ_BYTE=02) and (PIPE_FLAG=0)) then
         begin
              ERROR_FLAG:=2;
              Inc (PIPE_FLAG);
         end
      else if ((READ_BYTE=09) and (PIPE_FLAG=0)) then
      begin
       ERROR_FLAG:=1;
       Inc (PIPE_FLAG);
       Inc (PAKINERROR);
       GotoXY(5,22);
       write('ERROR PKT RECEIVED! NUMBER:=',PAKINERROR);
      end;
```

```
      Inc(PIPE_PTR);
   if (PIPE_PTR=32000) then PIPE_PTR:=0;

  if ERROR_FLAG=1 then
  begin
     UPDATE_ESDBAS;
  end;
  if (ERROR_FLAG=2) then
     begin
       UPDATE_NODELIST;
     end;
  end;
 goto 200;
500:SetIntVec($1C,Int1CSave);
CloseFile(dbas);
LeaveGraphic;
end.
```

{ The following is the source program listing of the compiled library (unit) named **USERINT** linked into the main program }

```
unit USERINT;
interface
uses Crt,Dos,Turbo3,Graph,Gdriver,Gwindow,Gkernel,Gshell,
Printer;
var
     MM:word;
     DD:word;
     YYYY:word;
     DOFW:word;
             Mth:string[2];
             Dy:string[2];
             Yr:string[4];
     code:integer;
             HOR:string[2];
             MINU:string[2];
             SECON:string[2];
             SCHUN:word;
    HR:word;
    MN:word;
    SC:word;

    p1,p2,p3,p4:pointer;
    size1,size2,size3,size4:word;
    PTR,COLNO,ROWNO,PRINT_I:integer;fname:string[16];chr,
                                                  Character:char;

     procedure DISPNETWORKSUMMARY;
     procedure DISPNODELISTSTAT;
     procedure DISPERRORSTAT;
     procedure INITDATETIME;
     procedure MENU;
     procedure MENU2;
     procedure DOSCOMMAND;
     procedure HELPSCREEN1;
     procedure HELPSCREEN2;
     procedure HELPSCREEN3;
     procedure OVER_ALL_HELP;
     procedure DEBUGROUTINEPROMPT;
     procedure CLEANSCREEN_VALUES;
implementation

procedure DISPNETWORKSUMMARY;{Routine #51}
begin
    ClearScreen;
    DefineWindow(1,0,0,XMaxGlb,YMaxGlb);
```

```
        DefineWorld(1,0,0,719,349);
        SelectWorld(1);
        SelectWindow(1);
        DrawBorder;
        DrawText(260,10,2,'NETWORK SUMMARY');
        SetLineStyle(0);
        GotoXY(5,3);
        write('       THROUGHPUT(in Bytes)');
        GotoXY(45,3);
        write('       UTILISATION');
        DrawStraight(0,719,28);
        DrawStraight(0,719,50);
        DrawText(10,60,2,'Current Previous Average Peak');
        DrawText(10,72,2,'Minute  Minute   per sec per Min ');
        DrawText(400,60,2,'Previous       Peak');
        DrawText(400,72,2,'Minute    ');
        DrawStraight(0,719,150);
        DrawLineClipped(393,28,393,285);
        GotoXY(12,12);
        write('DATA BASE SUMMARY');
        GotoXY(51,12);
        write('PACKET SUMMARY');
        DrawStraight(0,719,170);
        GotoXY(2,14);
        write('Sample Time(in min.):');
        GotoXY(2,16);
        write('# of Records in DBAS.DAT:');
        GotoXY(2,18);
        write('# of Records in NLDBAS.DAT:');
        GotoXY(2,20);
        write('# of Records in ESDBAS.DAT:');
        GotoXY(46,15);
        write('Packets in current minute:');
        GotoXY(46,17);
        write('Packets in Previous minute:');
        GotoXY(46,19);
        write('Average Number of packets:');
        GotoXY(46,20);
        write('in one second');
        MENU;
end;

procedure DISPNODELISTSTAT;{Routine #52}
begin
        ClearScreen;
        DefineWindow(2,0,0,XMaxGlb,YMaxGlb);
        DefineWorld(2,0,0,719,349);
        SelectWorld(2);
        SelectWindow(2);
```

```
        DrawBorder;
        DrawText(240,10,2,'NODE LIST STATISTICS');
        SetLineStyle(0);
        GotoXY(2,3);
        write('Connection ');
        GotoXY(2,4);
        write('Node');
        GotoXY(14,3);
        write('Bytes Xmitted');
        GotoXY(14,4);
        write('in previous');
        GotoXY(14,5);
        write('minute');
        GotoXY(29,3);
        write('Pkts. Xmitted');
        GotoXY(29,4);
        write('in previous');
        GotoXY(29,5);
        write('minute');
        GotoXY(45,3);
        write('Average');
        GotoXY(45,4);
        write('Pkt. Size');
        GotoXY(57,3);
        write('Pkts. rcvd.');
        GotoXY(57,4);
        write('in previous');
        GotoXY(57,5);
        write('minute');
        GotoXY(69,3);
        write('Bytes rcvd.');
        GotoXY(69,4);
        write('in previous');
        GotoXY(69,5);
        write('minute');
        DrawStraight(0,719,28);
        DrawStraight(0,719,70);
        DrawStraight(0,719,285);
        DrawLineClipped(112,28,112,285);
        DrawLineClipped(250,28,250,285);
        DrawLineClipped(370,28,370,285);
        DrawLineClipped(391,28,391,28);
        DrawLineClipped(487,28,487,285);
        DrawLineClipped(605,28,605,285);
        MENU;
end;
```

```
procedure DISPERRORSTAT;(Routine #53)
begin
    ClearScreen;
    DefineWindow(3,0,0,XMaxGlb,YMaxGlb);
    DefineWorld(3,0,0,719,349);
    SelectWorld(3);
    SelectWindow(3);
    DrawBorder;
    DrawText(260,10,2,'ERROR STATISTICS');
    SetLineStyle(0);
    DrawStraight(0,719,28);
    DrawStraight(0,719,285);
    GotoXY(3,4);
    write('TOTAL NUMBER OF PACKETS IN ERROR');
    GotoXY(43,4);
    write('NUMBER OF TOKEN LOSSES ');
    DrawStraight(0,719,63);
    DrawStraight(0,719,103);
    DrawLineClipped(359,28,359,63);
    GotoXY(3,6);
    write('Date');
    GotoXY(13,6);
    write('Time');
    GotoXY(25,6);
    write('Address of ');
    GotoXY(25,7);
    write('XMitting  Node');
    GotoXY(41,6);
    write('Address of ');
    GotoXY(41,7);
    write('Receiving Node');
    GotoXY(59,6);
    write('Length of ');
    GotoXY(59,7);
    write('Pkt.in Error');
    DrawLineClipped(88,63,88,285);
    DrawLineClipped(350,63,350,285);
    DrawLineClipped(200,63,200,285);
    DrawLineClipped(500,63,500,285);
    MENU;
end;

procedure MENU;(Routine #54)
const
    MaxWorldX:Float=719.0;
    MaxWorldY:Float=349.0;
var CharWidth,CharHeight:Float;
begin
  CharWidth := MaxWorldX / 80;
```

```
  CharHeight := MaxWorldY / 25;
  DrawSquare(5*CharWidth-2, 1*CharHeight-2,
             (13*CharWidth)+2, (2*CharHeight)+1, false);
  DrawSquare(64*CharWidth-2, 1*CharHeight-2,
             (75*CharWidth)+2, (2*CharHeight)+1, false);
  DrawStraight(0,719,285);
  DrawSquare(30,315,100,343,true);
  DrawSquare(130,315,200,343,true);
  DrawSquare(230,315,300,343,true);
  DrawSquare(330,315,400,343,true);
  DrawSquare(430,315,500,343,true);
  DrawSquare(530,315,600,343,true);
  DrawSquare(630,315,700,343,true);
  DrawStraight(0,719,308);
  GotoXY(2,24);
  write('F1');
  GotoXY(13,24);
  write('F2');
  GotoXY(24,24);
  write('F3');
  GotoXY(35,24);
  write('F4');
  GotoXY(46,24);
  write('F5');
  GotoXY(57,24);
  write('F6');
  GotoXY(68,24);
  write('F7');
  SetColorBlack;
  DrawText(35,323,1,'HELP');
  DrawText(135,323,1,'NETWORK');
  DrawText(135,333,1,'SUMMARY');
  DrawText(235,323,1,'NODE LIST');
  DrawText(235,333,1,'STATISTICS');
  DrawText(335,323,1,'ERROR');
  DrawText(335,333,1,'STATISTICS');
  DrawText(435,323,1,'GRAPHIC');
  DrawText(435,333,1,'DISPLAY');
  DrawText(535,323,1,'PRINT');
  DrawText(535,333,1,'SCREEN');
  DrawText(635,323,1,'OS Shell ');
  SetColorWhite;
end;
procedure MENU2;{Routine #55}
const
  MaxWorldX:Float=719.0;
  MaxWorldY:Float=349.0;
var CharWidth,CharHeight:Float;
begin
  CharWidth := MaxWorldX / 80;
```

```
  CharHeight := MaxWorldY / 25;
  DrawSquare(5*CharWidth-2, 1*CharHeight-2,
             (13*CharWidth)+2, (2*CharHeight)+1, false);
  DrawSquare(64*CharWidth-2, 1*CharHeight-2,
             (75*CharWidth)+2, (2*CharHeight)+1, false);
  DrawStraight(0,719,285);
  DrawSquare(30,315,100,343,true);
  DrawSquare(130,315,200,343,true);
  DrawSquare(230,315,300,343,true);
  DrawSquare(335,315,400,343,true);
  DrawSquare(435,315,500,343,true);
  DrawSquare(535,315,600,343,true);
  DrawSquare(630,315,700,343,true);
  DrawStraight(0,719,308);
  GotoXY(2,24);
  write('F5');
  GotoXY(13,24);
  write('F8');
  GotoXY(24,24);
  write('F9');
  GotoXY(35,24);
  write('F10');
  GotoXY(46,24);
  write('Sh');
  GotoXY(46,25);
  write('F5');
  GotoXY(57,24);
  write('Sh');
  GotoXY(57,25);
  write('F6');
  GotoXY(68,24);
  write('Sh');
  GotoXY(68,25);
  write('F1');
  SetColorBlack;
  DrawText(35,323,1,'THROUGHPUT');
  DrawText(35,333,1,'  GRAPH   ');
  DrawText(135,323,1,'PACKET');
  DrawText(135,333,1,'GRAPH');
  DrawText(235,323,1,'DATA BASE');
  DrawText(235,333,1,'RECORDS');
  DrawText(340,323,1,' VARIABLE');
  DrawText(340,333,1,' DUMP');
  DrawText(440,323,1,'GO TO');
  DrawText(440,333,1,'START');
  DrawText(540,323,1,'SHOW');
  DrawText(540,333,1,'SCREEN');
  DrawText(635,323,1,'QUIT');
  SetColorWhite;
end;
```

```
procedure INITDATETIME;{Routine #56}
 var
    Character:char;
    begin
      Delay(1000);
      GotoXY(1,4);
      writeln('Initialise Date and Time');
      GotoXY(1,6);
      writeln('Current Date is ');
      GotoXY(22,6);
      GetDate(YYYY,MM,DD,DOFW);
      writeln(MM,'-',DD,'-',YYYY);
      writeln('Do you want to change the Date ?(Y/N)');
      repeat
        Character:=Readkey;
        until ((Character='Y') or (Character='y') or
                     (Character='N') or (Character='n'));
        if (Character='Y') or (Character='y')  then
        begin
          GotoXY(1,10);
          repeat
            begin
               write('ENTER VALID MONTH(01-12):');
               Readln(Mth);
            end;
          until ((Mth >'00') and (Mth < '13'));

          begin
            Val(Mth,MM,code);
          end;
          repeat
            begin
              write('ENTER VALID DAY(01-31):');
              Readln(Dy);
            end;
          until ((Dy > '00') and (Dy < '32'));
          begin
            Val(Dy,DD,code);
          end;
          repeat
            begin
              write('ENTER VALID YEAR(1981-2099):');
              Readln(Yr);
            end;
          until ((Yr > '1980') and (Yr < '3000'));
          begin
            Val(Yr,YYYY,code);
          end;
        end;
        writeln;
```

```
        SetDate(YYYY,MM,DD);
        GetDate(YYYY,MM,DD,DOFW);
        writeln('Date Set:',MM,'-',DD,'-',YYYY);
        writeln;
        GetTime(HR,MN,SC,SCHUN);
        write('Current Time is  ',HR,':',MN,':',SC,':',
                                                 SCHUN);
        writeln;
        writeln('Do you want to set Time ?(Y/N)');
        repeat
          Character:=Readkey;
        until ((Character='Y') or (Character='y') or
                      (Character='N') or (Character='n'));
        writeln;
        writeln;
        if ((Character='Y') or (Character='y'))  then
        begin
          repeat
            write('ENTER VALID HOUR(00-23):');
            Readln(HOR);
            until ((HOR >='00') and (HOR < '24'));
            begin
              Val(HOR,HR,code);
            end;
            repeat
              write('ENTER VALID MINUTE(00-59):');
              Readln(MINU);
            until ((MINU >='00') and (MINU < '60'));
            begin
              Val(MINU,MN,code);
            end;
            repeat
              write('ENTER VALID SECOND(00-59):');
              Readln(SECON);
            until ((SECON >='00') and (SECON < '60'));
            begin
              Val(SECON,SC,code);
            end;
            SetTime(HR,MN,SC,00);
         end;
         GetTime(HR,MN,SC,SCHUN);
         writeln ('Time Set is  ',HR,':',MN,':',SC,':',
                                                 SCHUN);
         writeln;
         writeln('Press Key to Continue ......');
         repeat
         until keypressed;
      end;

procedure DOSCOMMAND;{Routine #57}
```

```
  var
    CURDIR:string[50];
    DRIVE:byte;
    extcde:word;
    begin
      LeaveGraphic;
      FillChar(CURDIR,SizeOf(CURDIR),' ');
      DRIVE:=0;
      GetDir(DRIVE,CURDIR);
      GotoXY(1,1);
      writeln('TYPE EXIT TO RETURN TO NETWORK MONITOR');
      writeln('CURRENT DIRECTORY=',CURDIR);
      writeln;
      writeln('NOTE:=DATA WILL BE LOST IF YOU DO NOT
                             RETURN  TO  MONITOR MODE ');
      writeln(' WITHIN ONE MINUTE.');
      Release(HeapOrg);
      Exec('\COMMAND.COM',' ');
      ChDir(CURDIR);
      EnterGraphic;
   end;

procedure HELPSCREEN1;(Routine #58)
  begin
     {Help for Network Summary Statistics}
end;
procedure HELPSCREEN2;(Routine #59)
begin
     {Help for Node List statistics}
end;
procedure HELPSCREEN3;(Routine #60)
  begin
      {Help for Error statistics}
  end;
procedure OVER_ALL_HELP;(Routine #61)
begin
      {Overall Help}
end;

procedure DEBUGROUTINEPROMPT;(Routine #62)
 begin
      ClearScreen;
      GotoXY(3,1);
      write('ASSEMBLY  ROUTINE VARIABLES');
      GotoXY(3,3);
      write('NO_OF_PACKETS_SEC:=');
      GotoXY(3,4);
      write('NO_OF_BYTES_SEC:=');
      GotoXY(3,6);
      write('INTM_BUF_PTR:=');
```

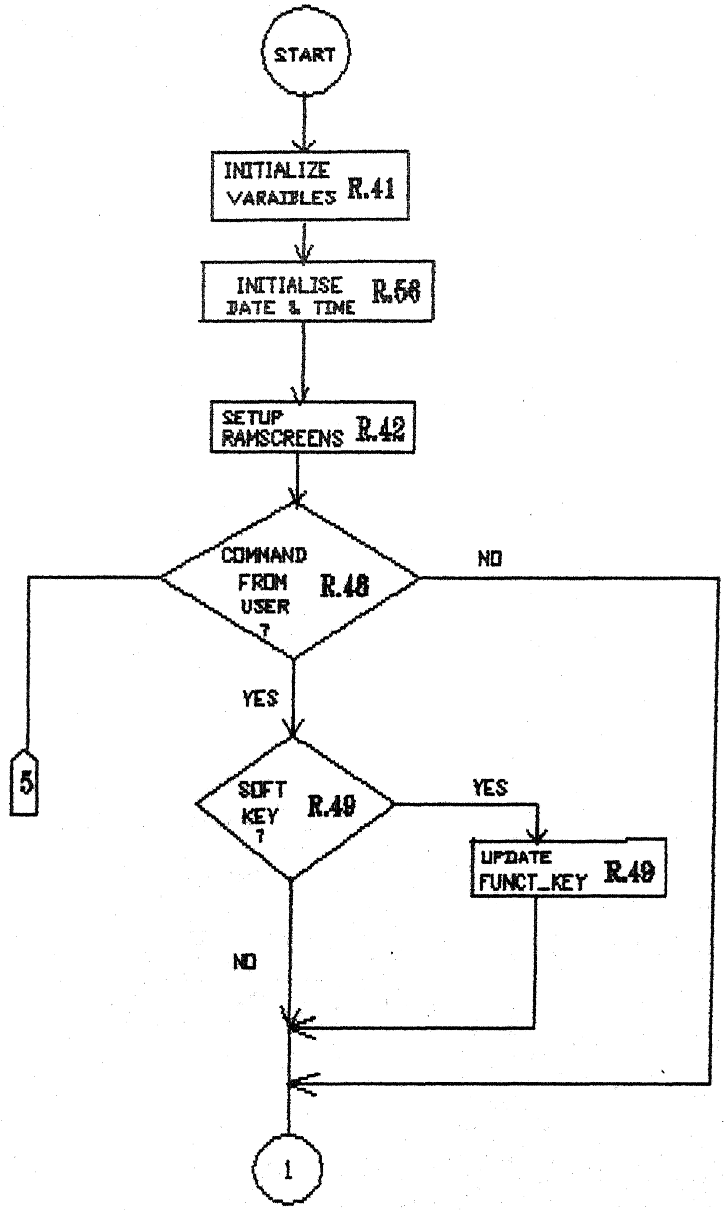
```
        GotoXY(3,7);
        write('THRPUT_INTM_BUF[INTM_BUF_PTR]:=');
        GotoXY(3,8);
        write('PAKSUM_INTM_BUF[INTM_BUF_PTR]:=');
        GotoXY(3,10);
        write('INITVARNETSUM VARIABLES');
        GotoXY(3,12);
        write('THRPUTCUR_MN');
        GotoXY(3,13);
        write('THRPUT_IN_PREV_MIN:=');
        GotoXY(3,14);
        write('THRPUT_AVG_MN:=');
        GotoXY(3,15);
        write('THRPUTPEK_OVER_HR:=');
        GotoXY(3,17);
        write('UTILIS_IN_PREV_MIN:=');
        GotoXY(3,18);
        write('UTILIS_AVG_MN:=');
        GotoXY(3,19);
        write('UTILISPEK_OVER_HR:=');
        GotoXY(3,21);
        write('PAKSUMCUR_MIN:=');
        GotoXY(3,22);
        write('PAKSUM_IN_PREV_MIN:=');
        GotoXY(3,23);
        write('PAKSUM_AVG_MN:=');
        GotoXY(3,24);
        write('PAKSUMPEK_OVER_HR:=');
        GotoXY(39,1);
        write('Variables of INITPROCESSECVAR');
        GotoXY(39,3);
        write('MATCH_INTM_BUF_PTR:=');
        GotoXY(39,10);
        write('NODE_LIST_PTR:= ');
        GotoXY(39,13);
        write('BUFF_PTR:= ');
        GotoXY(39,14);
        write('PIPE_PTR:=');
end;
procedure CLEANSCREEN_VALUES;{Routine #63}
begin
  GotoXY(32,3);
  write('        ');
  GotoXY(32,4);
  write('        ');
  GotoXY(32,6);
  write('         ');
  GotoXY(34,7);
  write('          ');
  GotoXY(34,8);
```

```
    write('        ');
    GotoXY(32,12);
    write('        ');
    GotoXY(32,13);
    write('        ');
    GotoXY(32,14);
    write('        ');
    GotoXY(32,15);
    write('        ');
    GotoXY(32,17);
    write('        ');
    GotoXY(32,18);
    write('        ');
    GotoXY(32,19);
    write('        ');
    GotoXY(32,21);
    write('        ');
    GotoXY(32,22);
    write('        ');
    GotoXY(32,23);
    write('        ');
    GotoXY(32,24);
    write('        ');
    GotoXY(60,3);
    write('        ');
    GotoXY(62,7);
    write('         ');
    GotoXY(62,8);
    write('         ');
  end;
  end.
```

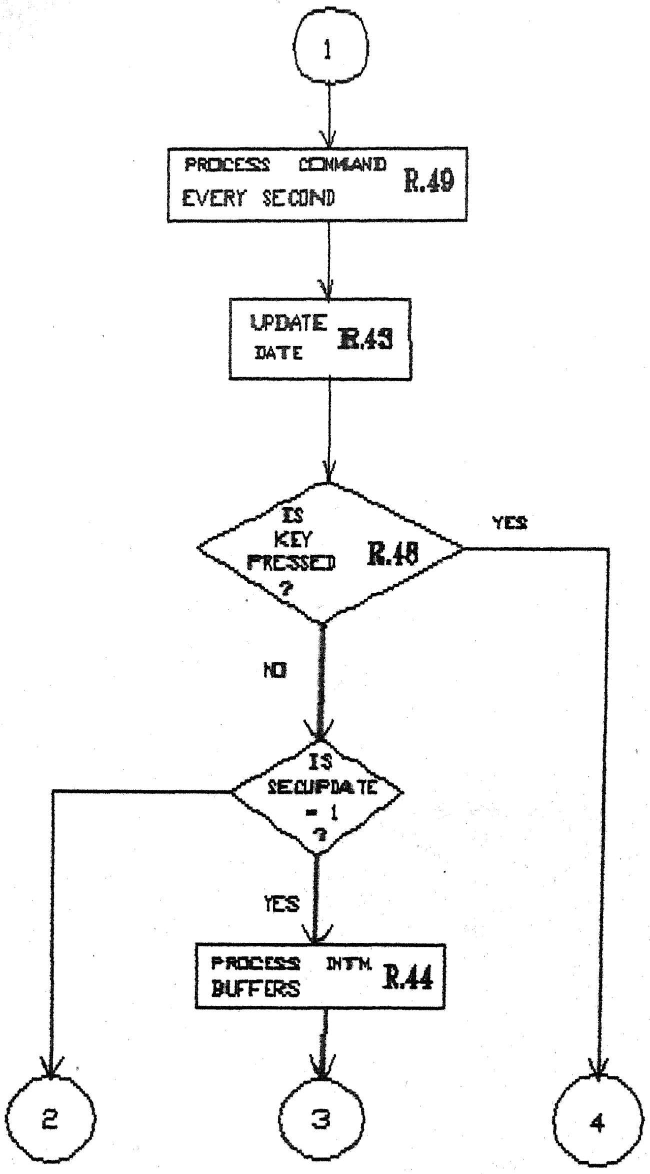
## INDEX TO ROUTINES IN THE LISTING OF MAIN PROGRAM

## INDEX TO ROUTINES IN THE LIBRARY(UNIT) USERINT

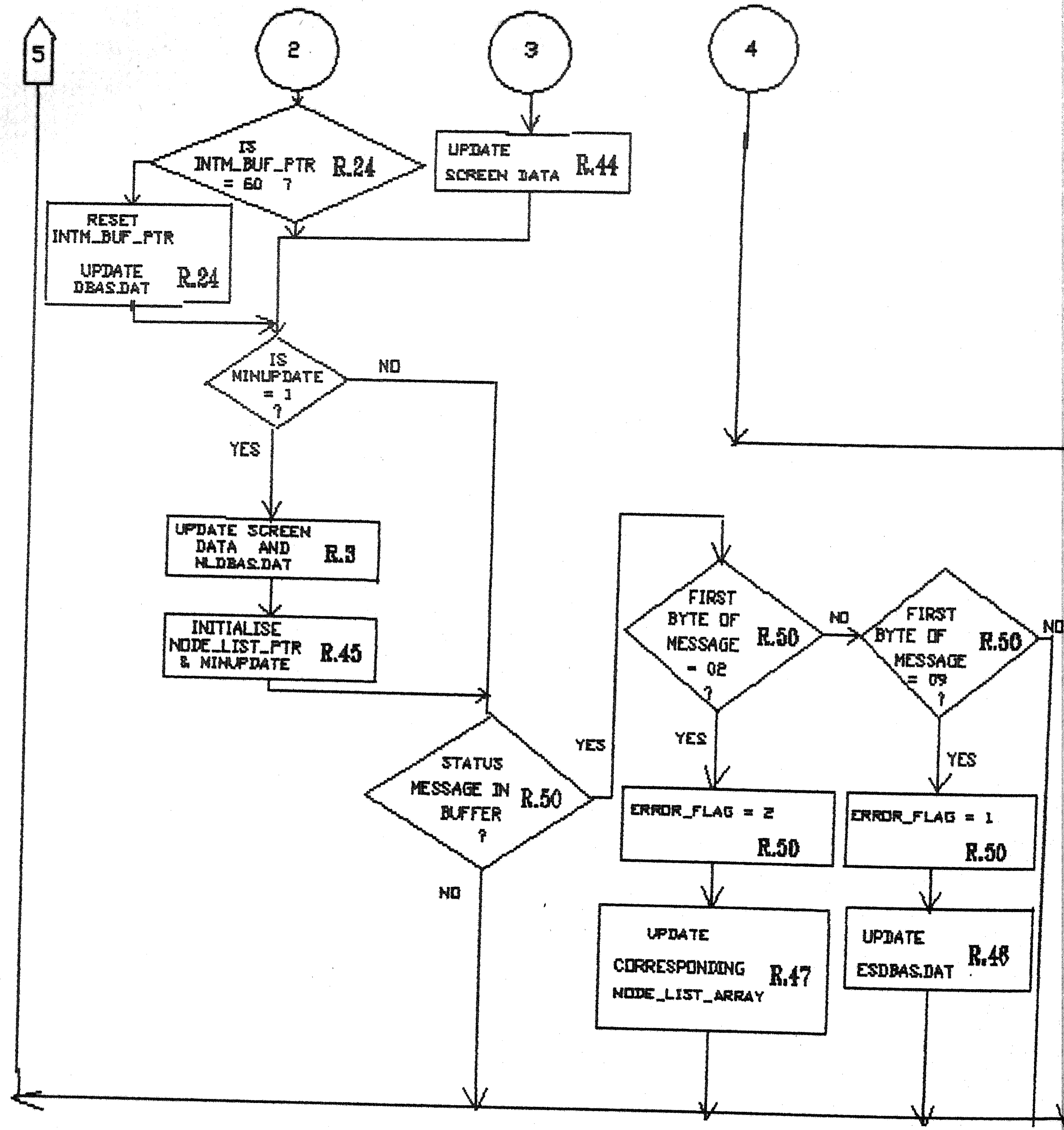
# FLOWCHART OF SOFTWARE IN PC




**NOTE:** **R.xx** refers to the number of the Routine in the program listing that performs the specified function. An Index to the Routines is given in pages 96 - 97.

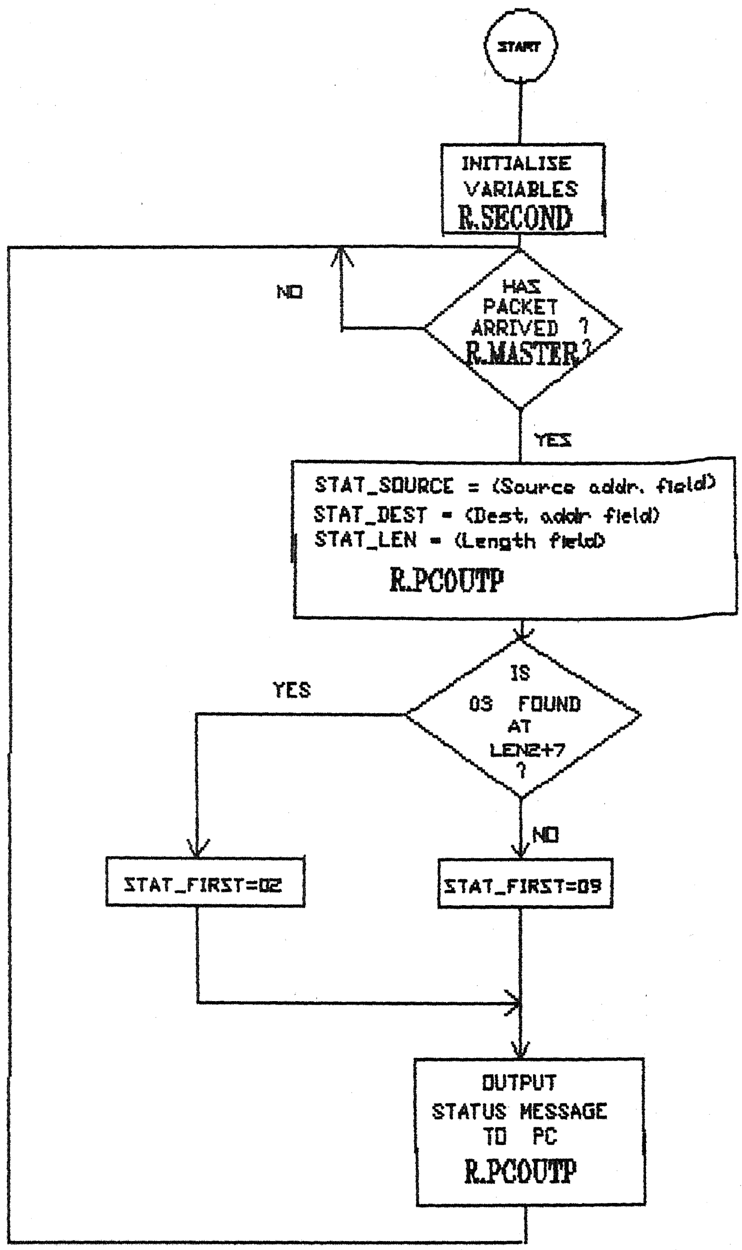



**NOTE:** **R.xx** refers to the number of the Routine in the program listing that performs the specified function. An Index to the Routines is given in pages 96 - 97.




**NOTE:** **R.xx** refers to the number of the Routine in the program listing that performs the specified function. An Index to the Routines is given in pages 96 - 97.

## FLOWCHART OF FIRMWARE




**NOTE:** R.xx refers to the name of the Routine in the program listing that performs the specified function.

# BIBLIOGRAPHY


[KP 87] Kornel Terplan, 'Communications Network Management', Prentice Hall Inc.

[JA 86] John Angermeyer, Kevin Jaeger, 'MS - DOS Developer's Guide', The Waite Group Co.

[JL 88] John Leong, 'A practical Guide to Ethernet', August 1988, Published in 'Tutorial, Local Network Technology' by William Stallings, IEEE Computer Society press

[SV 88] Sanjeev Verma, 'LOW COST IMPLEMENTATION OF PC LAN', A M.Tech Thesis submitted to IIT Kanpur


## GENERAL REFERENCES


1. Satyanarayanan M et. al., 'A Network Monitoring tool', Carnegie - Mellon University publication.

2. Information on Hewlett Packard LAN Analyser. (Model Nos. HP 4953A, HP 4954A, HP 4972A )

3. Information on Proteon's proNET Token Ring Monitor & Analyser. (Model Nos. p2302A & p2310)

4. Ulysses Black, 'Computer Networks - Protocols, Standards and Interfaces', Prentice Hall Inc.

5. Sudhakar G.N.M., Traffic Analysis on ALOHA, University of Hawaii

6. TURBO PASCAL 4.0, Owner's Handbook, Bourland International.

7. TURBO PASCAL GRAPHIX TOOLBOX, Owner's Handbook, Bourland International.

8. TURBO PASCAL DATABASE TOOLBOX, Owner's Handbook, Bourland International.

| | |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |